AFRL-RI-RS-TR-2016-242

CRITICAL NODE LOCATION IN DE BRUIJN NETWORKS

OCTOBER 2016

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2016-242 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
BRIAN ROMANO
Chief, Networking Technologies Branch
Computing & Communications Division

/ S /
JOHN MATYJAS
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
OCTOBER 2016	FINAL TECHNICAL REPORT	OCT 2014 – SEP 2016

4. TITLE AND SUBTITLE

CRITICAL NODE LOCATION IN DE BRUIJN NETWORKS

5a. CONTRACT NUMBER
IN-HOUSE: R182

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Victoria Goliber

5d. PROJECT NUMBER
HORN

5e. TASK NUMBER
IN

5f. WORK UNIT NUMBER
HO

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RITF
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITF
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2016-242

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2016-4782
Date Cleared: 3 OCT 2016

13. SUPPLEMENTARY NOTES

14. ABSTRACT

When deploying a wireless network, some highly desirable properties are (a) many short paths between any two nodes, and (b) relatively few edges. One type of network structure that satisfies both of these properties simultaneously is the class of de Bruijn networks. De Bruijn networks have been utilized in many applications, such as fault tolerant networks, peer-to-peer networks, amongst others. Because of their unique properties, many algorithms that are normally time-consuming perform exceptionally well on de Bruijn networks. This class of networks has yet to be considered from an identifying code perspective, and a complete examination of the problem is provided, from both a theoretical and algorithmic perspective.

15. SUBJECT TERMS

Identifying code; de Bruijn network; dominating set; resolving set; adiabatic quantum annealing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**VICTORIA GOLIBER**
U	U	U	UU	92	19b. TELEPHONE NUMBER (*Include area code*) **(315) 330-7703**

Table of Contents

List of Figures

Summary

The concept of identifying codes was originally developed as a means of pinpointing a specific critical node in a network, given its relationship with a special set of codeword nodes in a graph. Applications such as a fault diagnosis and sensor networks have found identifying codes extremely useful. For example, a network of smoke detectors with an accurate identifying code allows us to determine the exact location of a fire given only the set of detectors that have been triggered. Unfortunately, the problem of finding identifying codes is extremely computationally expensive, and so the real-world use so far has been minimal. To deal with this problem, we propose the use of a special network structure - de Bruijn networks.

When deploying a wireless network, some highly desirable properties are (a) many short paths between any two nodes, and (b) relatively few edges. One type of network structure that satisfies both of these properties simultaneously is the class of de Bruijn networks. De Bruijn networks have been utilized in many applications, such as fault tolerant networks, peer-to-peer networks, amongst others. Because of their unique properties, many algorithms that are normally time-consuming perform exceptionally well on de Bruijn networks. This class of networks has yet to be considered from an identifying code perspective, and a complete examination of the problem is needed, from both a theoretical and algorithmic perspective, and our initial theoretical results have shown promise.

Introduction

Consider a house with several different smoke detectors. In many cases, a fire will trigger not just one smoke detector, but several. Based on a specific set of detectors going off, can we accurately pinpoint the room in which there is a fire? This is an example of an identifying code. Each smoke detector has a certain radius that it covers, and some areas will be covered by more than one smoke detector. If each area of the house has a different set of detectors covering it, then the set of smoke detectors that go off completely determines where the fire is. With respect to sensor networks, if we can find a minimal identifying code for the network, then we can easily determine the location of the critical node.

In terms of graph theory, let G be an undirected graph. Let $B_t(v)$ be the ball of radius t around vertex v, i.e. the set of all vertices that are at distance at most t from v. A code is a set of vertices called codewords. Given a code S, the identifying set of a vertex v is $\mathsf{ID}_S(v) = B_t(v)S$. The code S is an identifying code if every identifying set in the graph is unique, or for vertices u, v we have $u = v$ if and only if $\mathsf{ID}_S(u) = \mathsf{ID}_S(v)$ [14].

While identifying codes have been considered for several specific types of graphs, they have yet to be examined for de Bruijn graphs. De Bruijn graphs have been useful in many applications. A de Bruijn graph of length n and alphabet d has a vertex for every string of length n over the set $\{0, 1, \ldots, d-1\}$. An edge is drawn starting at vertex $(u_1, u_2, u_3, \ldots, u_n)$ and ending at vertex $(v_1, v_2, v_3, \ldots, v_n)$ whenever $u_2 = v_1, u_3 = v_2, , u_n = v_{n-1}$. In other words, $(u_2, u_3, \ldots, u_n) = (v_1, v_2, \ldots, v_{n-1})$. We will refer to this graph as $\vec{\mathcal{B}}(d, n)$. Note that this definition produces a directed graph in which multiple edges and loops are allowed.

Definitions

3.1 General Graph Theory

Definition 3.1. The **distance** from vertex v to vertex u in a graph G is given by $d(u, v)$, and is defined as the length of the shortest path from u to v in G. If G is a digraph, then we require this path to be a directed path. We define $d(u, u) = 0$.

Definition 3.2. Two vertices $u, v \in V(G)$ are **adjacent** if either $d(u, v) = 1$ or $d(v, u) = 1$. We denote this $u \sim v$.

Definition 3.3. Let $v \in V(G)$. The **open in-neighborhood** of v is given by $N^-(v) = \{u \in V(G) \mid d(u, v) = 1\}$, and the **closed in-neighborhood** is given by $N^-[v] = N^-(v) \cup \{v\}$. The **open out-neighborhood** is given by $N^+(v) = \{u \in V(G) \mid d(v, u) = 1\}$, and the **closed out-neighborhood** of vertex v is given by $N^+[v] = N^+(v) \cup \{v\}$. In an undirected graph, an **open neighborhood** of v is $N(v) = \{u \in V(G) \mid d(u, v) = 1\}$ and the **closed neighborhood** of v is $N[v] = N(v) \cup \{v\}$.

Definition 3.4. The **in-ball of radius** t centered at vertex v is the set: $B_t^-(v) = \{u \in V(G) \mid d(v, u) \leq t\}$. and the **out-ball of radius** t centered at vertex v is the set: $B_t^+(v) = \{u \in V(G) \mid d(u, v) \leq t\}$. In an undirected graph, the **ball of radius** t centered at vertex v is the set $B_t(v) = \{u \in V(G) \mid d(v, u) \leq t\}$.

Definition 3.5. Two vertices $u, v \in V(G)$ are called **t-twins** if $B_t^-(u) = B_t^-(v)$. If the graph has no t-twins, then G is called **t-twin-free**. For an undirected graph, we use the same definition with in-ball replaced with ball.

Definition 3.6. Given a subset $S \subset V(G)$, the S **t-identifying set** for vertex v is given by $\mathsf{ID}_S(v) = B_t^-(v) \cap S$. For an undirected graph, we use the same definition with t-in-ball replaced with t-ball.

3.2 Types of Identifying Sets

Definition 3.7. A **t-dominating set** is a set $S \subseteq V(G)$ such that for all $v \in V(G)$ we have $B_t^-(v) \cap S \neq \emptyset$. This is equivalent to saying that $\bigcup_{v \in S} B_t^+(v) =$

$V(G)$. For an undirected graph, we replace $B_t^-(v)$ and $B_t^+(v)$ with $B_t(v)$. The **t-domination number**, denoted $\mathsf{dom}^t(G)$, is the minimum size of a t-dominating set in G.

Definition 3.8. [1] A **k-tuple dominating set** of a graph is a subset S of vertices such that every vertex is 1-dominated by at least k vertices in S. The **k-tuple domination number**, denoted $\gamma_{\times k}(G)$ is the minimum size of a k-tuple dominating set in G.

Definition 3.9. A **distinguishing set** is a set S of vertices such that for all pairs of vertices $u, v \in V(G)$ we have either:

1. $u \in S$, or
2. $v \in S$, or
3. $N(u) \cap S \neq N(v) \cap S$.

Definition 3.10. A **locating-dominating set** is a set S of vertices such that for all pairs of vertices $u, v \in V(G)$ we have either:

1. $u \in S$, or
2. $v \in S$, or
3. $B_t^-(u) \cap S \neq B_t^-(v) \cap S$.

For an undirected graph, replace t-in-ball with t-ball.

Definition 3.11. A **t-identifying code** is a t-dominating set $S \subseteq V(G)$ such that for all pairs $u, v \in V(G)$ we have $\mathsf{ID}_S(u) \neq \mathsf{ID}_S(v)$. (Note that since S is a t-dominating set, we are also requiring that $\mathsf{ID}_S(x) \neq \emptyset$ for all $x \in V(G)$.) The variable t is referred to as the **radius** of the identifying code. We denote the size of a minimum identifying code by $\gamma^{\mathsf{ID}}(G)$.

Some authors will allow a t-identifying code to admit at most one non-empty identifying set. Unless otherwise specified, will require every identifying set to be nonempty.

Definition 3.12. A **k-robust t-identifying code** is a t-identifying code $S \subseteq V(G)$ such that removal of any set $T \subseteq S$ with $|T| \leq k$ preserves the t-identifying properties, i.e. $S \setminus T$ is a t-identifying code.

Definition 3.13. [8] A **directed resolving set** is a set S so that for each $v \in V(G)$ there exist $u_1, u_2 \in S$ so that $d(v, u_1) \neq d(v, u_2)$. The **directed metric dimension** is the minimum size of a directed resolving set.

Definition 3.14. [6] A **determining set** or **fixing set** is a set S so that the only automorphism that fixes the vertices of S pointwise is the trivial automorphism.

Note that an alternate definition for a determining set is a set S for which whenever $f, g \in \mathrm{Aut}(G)$ so that $f(s) = g(s)$ for all $s \in S$, then $f(v) = g(v)$ for all $v \in V(G)$. That is, every automorphism is completely determined by its action on a determining set.

In the above definitions, if t is omitted from the notation (i.e. identifying code instead of t-identifying code), then it is assumed that $t = 1$. Note also that these definitions have corresponding counterparts for undirected graphs.

3.3 Strings and the Directed de Bruijn Graph

We will be considering various types of vertex subsets on the class of directed de Bruijn graphs. The following definitions will be useful in working with this class of graphs. We will use the notation $[x] = \{1, 2, \ldots, x\}$.

Definition 3.15. Let $\boldsymbol{\mathcal{A}_d} = \{0, 1, \ldots, d-1\}$ and let $\mathcal{A}_d^n$ be the set of all strings of length n made up of **letters** of $\mathcal{A}$. When d is clear from context we will use $\mathcal{A}$ and $\mathcal{A}^n$ respectively.

Definition 3.16. The **directed de Bruijn graph**, denoted $\boldsymbol{\vec{\mathcal{B}}(d, n)}$, has vertex set $\mathcal{A}_d^n$. An edge from vertex $x_1x_2\ldots x_n$ to vertex $y_1y_2\ldots y_n$ exists if and only if $x_2x_3\ldots x_n = y_1y_2\ldots y_{n-1}$.

Definition 3.17. The **concatenation of two strings** $x = x_1x_2\ldots x_i$ and $y = y_1y_2\ldots y_k$ is given by $x \oplus y = x_1x_2\ldots x_iy_1y_2\ldots y_k$.

Definition 3.18. The **concatenation of sets of strings** S and T is given by $S \oplus T = \{x \oplus y \mid x \in S \text{ and } y \in T\}$.

Definition 3.19. The **prefix** of a string $x = x_1x_2\ldots x_n$ is the substring $x_1x_2\ldots x_{n-1}$, denoted by $\boldsymbol{x^-}$.

Definition 3.20. The **suffix** of a string $x = x_1x_2\ldots x_n$ is is the substring $x_2x_3\ldots x_n$, denoted by $\boldsymbol{x^+}$.

Definition 3.21. When discussing substrings of a string $x_1x_2\ldots x_n$, we will use the notation $\boldsymbol{x(a:b)}$ to denote the substring $x_ax_{a+1}\ldots x_b$.

Definition 3.22. If a string $x = x_1x_2\ldots x_n$ contains a constant substring $x(a, b) = zz\ldots z$, then we will denote the consecutive letters as z^{b-a}, the constant raised to the power denoting length. This will also be used for repeated substrings, such as $0101\ldots01 = (01)^k$.

Definition 3.23. Let $w = w_1\ldots w_n \in \mathcal{A}_d^n$. Define $w^{(t,m)} = w_1^{(t,m)}\ldots w_n^{(t,m)}$ such that:

$$w_i^{(t,m)} = \begin{cases} w_t + m \pmod d, & \text{if } i = t; \\ w_i, & \text{otherwise.} \end{cases}$$

Definition 3.24. Let $w = w_1\ldots w_n \in \mathcal{A}_d^n$ and $\ell \in \mathbb{Z}^+$ such that $n \geq 2\ell$. Then we say that w has *period length* ℓ if $w_i = w_{i+\ell}$ for all $i \in [n-\ell]$. If we have $n < 2\ell$, then we say that w has *almost period length* ℓ.

Definition 3.25. Let $w \in \mathcal{A}_d^n$, and suppose that w has period length ℓ, and does not have period length k for any $k < \ell$. Then w is called *ℓ-periodic*.

Definition 3.26. Let $w \in \mathcal{A}_d^n$. If there exists some $\ell > \frac{n}{2}$ and word $w' \in \mathcal{A}_d^{2\ell-n}$ such that $w \oplus w'$ is ℓ-periodic, then w is called *almost ℓ-periodic*.

We now provide some lemmas regarding string properties that will be used later in the paper.

Lemma 3.27. *[10] Let $\ell_1 > \ell_2$ and w be a word of length $n \geq \ell_1 + \ell_2 - gcd(\ell_1, \ell_2)$. If w has periods (or almost periods) of length ℓ_1 and ℓ_2 , then w has a period of length $gcd(\ell_1, \ell_2)$.*

Lemma 3.28. *[4] Let $\ell_1 \geq \ell_2$ and w be a word of length $n \geq \ell_1 + \ell_2$. If w has a period (or almost period) of length ℓ_1 and $w^{(k,m)}$ has a period of length ℓ_2 for some $m \in \mathcal{A}_d$, then there is $m' \in \mathcal{A}_d$ such that $w^{(k,m')}$ has a period of length $gcd(\ell_1, \ell_2)$.*

Lemma 3.29. *Let $w \in \mathcal{A}_d^n$ such that w is ℓ_1-periodic or almost ℓ_1-periodic. Let $m \in [d-1]$ and also $k \in [n]$ with $k \leq n - \ell_1$ or $k > \ell_1$. Then for any $\ell_2 < \frac{n}{2}$ with $\ell_1 \geq \ell_2$ and $n \geq \ell_1 + \ell_2$, it is not possible that $w^{(k,m)}$ is ℓ_2-periodic.*

Proof. We proceed by contradiction, and suppose that $w^{(k,m)}$ is ℓ_2-periodic. We have two cases. First, if $k > \ell_1$, then by Lemma 3.28, there exists some $m' \in \mathcal{A}_d$ such that $w^{(k,m')}$ has period of length $\gcd(\ell_1, \ell_2)$. Then we have the following chain of equalities.

$$\begin{aligned} w_k &= w_{k-\ell_1} && \text{since } w \text{ has a period of length } \ell_1 \\ &= w_{k-\ell_2} && \text{since } w^{(k,m')} \text{ has a period of length } \gcd(\ell_1, \ell_2) \\ &= w_k^{(k,m)} && \text{since } w^{(k,m)} \text{ has a period of length } \ell_2 \end{aligned}$$

Hence this is a contradiction. For our second case, when $k \leq n - \ell_1$, we note the following.

$$w_k = w_{k+\ell_1} = w_{k+\ell_2} = w_k^{(k,m)}$$

This is also a contradiction. Therefore we must have that $w^{(k,m)}$ is not ℓ_2-periodic. □

Lemma 3.30. *Let $w \in \mathcal{A}_d^n$ such that w has period length ℓ_1 for some $\ell_1 < \frac{n}{2}$. For all $m \in [d-1]$ and for all $i, j, k \in [n]$ with $i \leq k \leq j$, and for all $\ell_2 \leq \ell_1$ with $j - i + 1 \geq \ell_1 + \ell_2$ and with either $k \geq i + \ell_1$ or $k \leq j - \ell_1$, we must have that $w^{(k,m)}(i, j)$ does not have period ℓ_2.*

Proof. Define $w' = w(i, j)$ and $w'^{(k-i,m)} = w^{(k,m)}(i, j)$, and then apply Lemma 3.29 to compare the two strings. □

Lemma 3.31. *Let $n = 2t$ and let $u \in \mathcal{A}_d^n$. If u has period length t and for some $\ell < t$ and $m \in \mathcal{A}_d$, we find that $u' = u^{(t,m)}(t + 1 - \ell : n - 1)$ is ℓ-periodic, then we must have that ℓ divides t and $u' \oplus (u_n + m)$ has period ℓ.*

Proof. First, we note that $U_m = \left(u^{(t,m)}\right)^{(n,m)}$ clearly has period length t, and so $U_m(t+1-\ell : n-1)$ has almost period length t. Additionally, since $U_m(t+1-\ell : n-1) = u'$, we know that $U_m(t+1-\ell : n-1)$ is ℓ-periodic. Hence by Lemma 3.27, $U_m(t+1-\ell : n-1)$ has period of length $p = \gcd(t, \ell)$. However since u' is given to be ℓ-periodic, the minimum period length is ℓ and so we must have that $p = \ell$ and thus ℓ divides t.

To show that $u' \oplus (u_n + m)$ has period ℓ, we note that u' has period ℓ, and that $u'_\ell = u_t^{(t,m)} = u_t + m$. Having period ℓ implies that $u'_k = u_t + m$ for all k that is divisible by ℓ. Since $u_n + m$ is the $(t+\ell)$th letter in $u' \oplus (u_n + m)$, and this is divisible by ℓ, we need that $u_n + m = u_t + m$ in order for $u' \oplus (u_n + m)$ to have period ℓ. But this is given to be true since u has period t. □

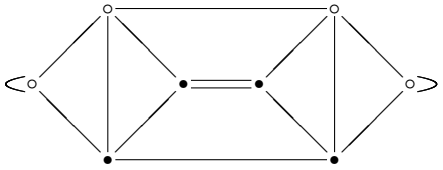
The two following lemmas are useful in working with distances in $\vec{\mathcal{B}}(d, n)$ and their proofs are self-evident.

Lemma 3.32. In $\vec{\mathcal{B}}(d, n)$ there is a directed path of length $t \leq n$ from x to y if and only if $x(t+1 : n) = y(1 : n-t)$. That is, if and only if the rightmost $n-t$ letters of x are the same as the leftmost $n-t$ letters of y.

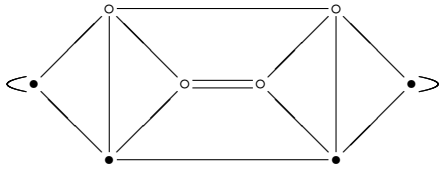
Lemma 3.33. In $\vec{\mathcal{B}}(d, n)$ if vertices $x \neq y$ have the same prefix, then for all $u \neq \{x, y\}, \vec{d}(u, x) = \vec{d}(u, y)$. In particular, $B_t^-(x) \setminus \{x\} = B_t^-(y) \setminus \{y\}$ for all $t \leq n$.

3.4 Examples

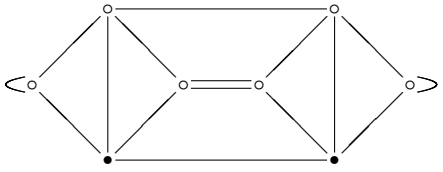
Identifying Code



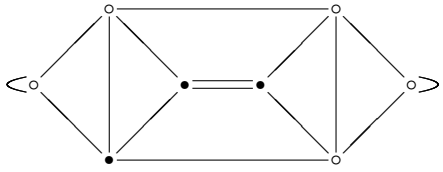
Locating-Dominating Set



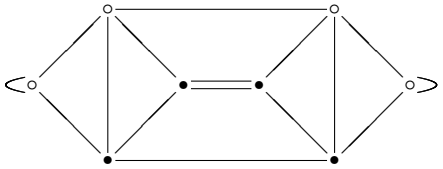
Dominating Set



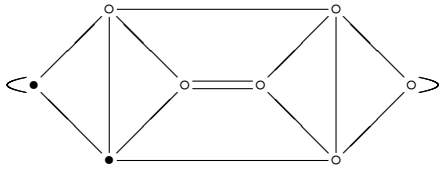
Distinguishing Set



Resolving Set



Determining Set



3.5 de Bruijn Results

In this section, we list some general results concerning the de Bruijn graph that were determined in our search for special vertex sets.

Lemma 3.34. *The strings in $B_t(x)$ for $x = x_1x_2\dots x_n$ must be one of the following three types.*

1. *x;*
2. *$[d]^g \oplus x_{b-f+1}\dots x_{n-f} \oplus [d]^{b-g}$ with $b > f, b > g, f + b + g \leq t$;*
3. *$[d]^{f-c} \oplus y_{b+1}\dots y_{n-f+b} \oplus [d]^c$ with $f > b, f > c, b + f + c \leq t$.*

Proof. All strings in $B_t(x)$ can be described by following forward or backward edges. The strings of type (1) are reached by taking no moves. All other strings (types (2) and (3)) are reached by taking either moves of type FBF (forward-backward-forward) or BFB (backward-forward-backward). We will describe *shortest* paths within these confines. We define f steps forward from vertex $x_1x2\dots x_n$ as reaching vertices in the set:

$$[d]^f \oplus x_1\dots x_{n-f}.$$

We define b steps backward from vertex $x_1x_2\dots x_n$ as reaching vertices in the set :

$$x_{b+1}\dots x_n \oplus [d]^b.$$

If FBF is the shortest path to reach some vertex y from x, then we must follow f edges forward, b edges backward, and g edges forward, with the constraints that $b > f$, $b > g$, and $f + b + g \leq t$. Following these sequences, we arrive at strings of type (2).

If BFB is the shortest path to reach some vertex y from x, then we must follow b edges backward, f edges forward, and c edges backward, with the constraints that $f > b$, $f > c$, and $b + f + c \leq t$. Following these sequences, we arrive at strings of type (3). □

Lemma 3.35. *For any $y \in \mathcal{B}(d,n)$ with $d \geq 3$, there exists some vertex x such that $d(y,x) = n$.*

Proof. We proceed by induction on n and show that if the claim is true in $\mathcal{B}(d,n)$ for $n \geq 2$, then the claim is true for $\mathcal{B}(d,n+2)$.

Base Case: $n = 2$. Since $d \geq 3$, our vertex $y = y_1y_2$ can use at most two symbols from our alphabet. Suppose that $z \in [d]\setminus\{y_1,y_2\}$. Then $d(y,zz) = 2$.

As our induction proceeds from string length n to $n+2$, we require an additional base case of $n = 3$. If our vertex $y = y_1y_2y_3$ only uses two distinct symbols from $[d]$, then the string $x = a^n$ where $a \in [d]\setminus\{y_1,y_2,y_3\}$ satisfies $d(y,x) = 3$. Otherwise, we must have $[d] = \{y_1,y_2,y_3\}$. Then the vertex $x = (y_2)^3$ satisfies $d(y,x) = 3$.

Induction Step: Let $\overline{y} = y_0 \oplus y \oplus y_{n+1}$ be arbitrary. By the induction hypothesis, there exists some $x \in \mathcal{B}(d,n)$ such that $d(x,y) = n$. We will show that $d(\overline{y},\overline{x}) = n+2$, where $\overline{x} = x_0 \oplus x \oplus x_{n+1}$ with $x_0 \in [d] \setminus \{y_n, y_{n+1}\}$ and $x_{n+1} \in [d] \setminus \{y_0, y_1\}$. We will show that $\overline{x} \notin B_{n+1}(y)$ using Lemma 3.34 and considering each type of path and resulting string individually.

1. $\overline{x} = \overline{y}$. Not possible since $x \neq y$.
2. FBF-type.
   First, from Lemma 3.34, we know that since $d(x,y) = n$ there cannot exist any choice of f,b,g such that $f + b + g \leq n-1$, $b > 0$, $b > f$, and $b > g$ such that

   $$x \in [d]^g \oplus y_{b-f+1}\dots y_{n-f} \oplus [d]^{b-g}.$$

   In other words, we must have

   $$y_{b-f+1}\dots y_{n-f} \neq x_{g+1}\dots x_{g+n-b}$$

   for all such choices of f,b,g.
   Now we will show that there does not exist an FBF-path of length $n+1$ or less between $\overline{x}$ and $\overline{y}$. Fix some f,b,g such that $f + b + g \leq$

$n+1$, $b > 0$, $b > f$, and $b > g$. From Lemma 3.34 all vertices $z_0 z_1 \dots z_{n+1}$ that can be reached by an FBF-path with parameters (f, b, g) from $\overline{y}$ must have

$$y_{b-f} \dots y_{n-f+1} = z_g \dots z_{g+n+1-b}.$$

(a) If $f = 0$, $b = k$, and $g = 0$, then we consider $1 \leq k \leq n-1$ and $n \leq k \leq n+1$ separately. First, if $1 \leq k \leq n-1$, then our induction hypothesis with parameters $(0, k, 0)$ tells us that $x_1 \dots x_{n-k} \neq y_{k+1} \dots y_n$ when we examine FBF-paths with parameters $(0, k, 0)$ from y. Hence we cannot have $x_0 \dots x_{n-k+1} = y_k \dots y_{n+1}$, and so no such FBF-path exists between $\overline{x}$ and $\overline{y}$. Next, if $n \leq k \leq n+1$, then since $x_0 \neq y_n, y_{n+1}$, we will never have $x_0 x_1 = y_n y_{n+1}$ or $x_0 = y_{n+1}$, and so again no such FBF-path exists in $\mathcal{B}(d, n+2)$.

(b) If $f \geq 1$, then we have $b \geq 2$. In this case, in order for such an FBF-path to exist from $\overline{y}$ to $\overline{x}$ we need $x_g \dots x_{g+n-b+1} = y_{b-f} \dots y_{n+1-f}$. However our induction hypothesis with parameters $(f-1, b-1, g)$ shows $x_{g+1} \dots x_{g+n-b+1} \neq y_{b-f+1} \dots y_{n-f+1}$, and so no such FBF-path exists in $\mathcal{B}(d, n+2)$.

(c) If $g \geq 1$, then again we must have $b \geq 2$. In this case, in order for such an FBF-path to exist we must have $x_g \dots x_{g+n-b+1} = y_{b-f} \dots y_{n+1-f}$. However our induction hypothesis with parameters $(f, b-1, g-1)$ tells us that $x_g \dots x_{g+n-b} \neq y_{b-f} \dots y_{n-f}$, and so no such FBF-path exists in $\mathcal{B}(d, n+2)$.

Hence we cannot have an FBF-path of length less than $n+2$ between $\overline{y}$ and $\overline{x}$ in $\mathcal{B}(d, n+2)$.

3. BFB-type.

First, from Lemma 3.34, we know that since $d(x, y) = n$ there cannot exist any choice of b, f, c such that $b + f + c \leq n-1$, $f > 0$, $f > b$, and $f > c$ such that

$$x \in [d]^{f-c} \oplus y_{b+1} \dots y_{n-f+b} \oplus [d]^c.$$

In other words, we must have

$$y_{b+1} \dots y_{n-f+b} \neq x_{f-c+1} \dots x_{n-c}$$

for all such choices of b, f, c.

Now we will show that there does not exist a BFB-path of length $n+1$ or less between $\overline{x}$ and $\overline{y}$. Fix some b, f, c such that $b + f + c \leq n+1$, $f > 0$, $f > b$, and $f > c$. From Lemma 3.34 all vertices $z_0 z_1 \dots z_{n+1}$ that can be reached by a BFB path from $\overline{y}$ with these parameters must have

$$y_b \dots y_{n+1-f+b} = z_{f-c} \dots z_{n+1-c}.$$

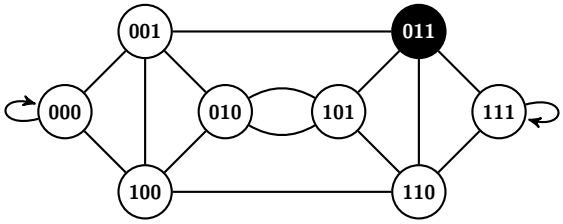


Figure 1: $\mathcal{B}(2,3)$ does not contain any vertices at distance 3 from 011.

(a) If $b = 0$, $f = k$, and $c = 0$, then we consider $1 \leq k \leq n-1$ and $n \leq k \leq n+1$ separately. First, if $1 \leq k \leq n-1$, then our induction hypothesis tells us that $x_{k+1} \dots x_n \neq y_1 \dots y_{n-k}$ when we examine BFB-paths with parameters $(0, k, 0)$ from y. Hence we cannot have $x_k \dots x_{n+1} = y_0 \dots y_{n-k+1}$ in $\mathcal{B}(d, n+2)$, so no such BFB-path exists between $\overline{x}$ and $\overline{y}$.
Next, if $n \leq k \leq n+1$, then since $x_{n+1} \neq y_0, y_1$, we will never have $x_n x_{n+1} = y_0 y_1$ or $x_{n+1} = y_0$, and so again no such BFB-path exists in $\mathcal{B}(d, n+2)$.

(b) If $b \geq 1$, then we have $f \geq 2$. In this case, in order for such a BFB-path to exist from $\overline{x}$ to $\overline{y}$ we must have $x_{f-c} \dots x_{n+1-c} = y_b \dots y_{n+1-f+b}$. However our induction hypothesis with parameters $(b-1, f-1, c)$ tells us that $x_{f-c} \dots x_{n-c} \neq y_b \dots y_{n-f+b}$, and so no such BFB-path exists in $\mathcal{B}(d, n+2)$.

(c) If $c \geq 1$, then we must have $f \geq 2$. In this case, to have such a BFB-path between $\overline{x}$ and $\overline{y}$ we must have $x_{f-c} \dots x_{n+1-c} = y_b \dots y_{n+1-f+b}$. However our induction hypothesis with parameters $(b, f-1, c-1)$ shows $x_{f-c+1} \dots x_{n-c+1} \neq y_{b+1} \dots y_{n-f+1+b}$, and so no such BFB-path exists in $\mathcal{B}(d, n+2)$.

Hence we cannot have a BFB-path of length less than $n+2$ between $\overline{y}$ and $\overline{x}$ in $\mathcal{B}(d, n+2)$.

Therefore there is no path from $\overline{y}$ to $\overline{x}$ of length $n+1$ or smaller, and so $d(\overline{y}, \overline{x}) \geq n+2$. As it is well known that the de Bruijn graph $\mathcal{B}(d, n+2)$ has diameter $n+2$ (see [2]), we must have $d(\overline{y}, \overline{x}) = n+2$.

□

In other words, Lemma 3.35 tells us the eccentricity of every node in the graph $\mathcal{B}(d, n)$ is n for $d \geq 3$, and so the radius of $\mathcal{B}(d, n)$ is n. Note that when $d = 2$ this does not always hold. For example, the graph $\mathcal{B}(2, 3)$ does not have any vertex at distance 3 from 011. See Figure 1.

Results and Discussion

4.1 Dominating Set Bounds

First, we recall the definition of a dominating set in a graph.

Definition 4.36. A (directed) t**-dominating set** is a subset $S \subseteq V(G)$ such that for all $v \in V(G)$ we have $B_t^-(v) \cap S \neq \emptyset$. That is, S is a (directed) t-dominating set if every vertex in G is within (directed) distance t of some vertex in S. We denote the size of a minimum t-dominating set in a graph G by $\gamma_t(G)$.

Note that by definition every identifying code is also a dominating set, but not conversely.

4.1.1 Directed de Bruijn Graphs

We begin with a review of the current literature and then proceed with our results.

Theorem 4.37. *[15] For $d \geq 2$, $n \geq 1$, $\gamma_1(\vec{\mathcal{B}}(d,n)) = \left\lceil \frac{d^n}{d+1} \right\rceil$.*

In [15] a construction of a minimum dominating set for $\vec{\mathcal{B}}(d,n)$ is given. Key to this construction is the fact that every integer m corresponds to a string (base d) in $\mathbb{Z}_d^n$, that we call X_m. The construction utilizes a special integer m defined by:

$$m = \begin{cases} d^{n-2} + d^{n-4} + \cdots + d^{n-2k} + \cdots + d^2 + 1 \bmod d^n, & \text{if } n \text{ is even;} \\ d^{n-2} + d^{n-4} + \cdots + d^{n-2k} + \cdots + d^3 + d \bmod d^n, & \text{if } n \text{ is odd.} \end{cases}$$

Let $D = \{m, m+1, \ldots, m + \lceil \frac{d^n}{d+1} \rceil - 1\}$. Now let S be the set of strings $\{X_i \mid i \in D\}$. Then S is a minimum size dominating set for $\vec{\mathcal{B}}(d,n)$.

Next we provide constructions for t-dominating sets. While others have considered some variations of t-dominating sets (such as perfect dominating sets in [18]), it does not appear that the general t-dominating sets have been considered in the directed de Bruijn graph.

Theorem 4.38. *The set* $S \cup \{0^n\}$ *where*

$$S = \{x \in \mathcal{A}_d^n \,|\, x_{k(t+1)} \neq 0 \text{ for some } k \in \mathbb{Z}_+ \text{ and } x_i = 0 \text{ for all } i < k(t+1)\}$$

is a t*-dominating set of size*

$$1 + d^{n-t-1}(d-1)\left(\frac{1 - d^{-(t+1)\left\lfloor \frac{n}{t+1} \right\rfloor}}{1 - d^{-(t+1)}}\right)$$

in $\vec{\mathcal{B}}(d, n)$.

Proof. Let x be a vertex in $\mathcal{A}_d^n$. Assume that there are k zeros at the beginning of x, but not $k+1$ zeros, i.e. $x = 0^n$ or $x = 0^k \oplus a \oplus x(k+2 : n)$ for some $a \neq 0$. Let $l \in [0, t]$ be an integer so that $k + l \equiv t \pmod{t+1}$, i.e. $k + l = t + m(t+1) = (m+1)(t+1) - 1$ for some $m \geq 0$. Now

$$0^l \oplus x(1 : n-l) = 0^l \oplus 0^k \oplus a \oplus x(k+2 : n-l) = 0^{k+l} \oplus a \oplus x(k+2 : n-l)$$

belongs to S except if $k+l \geq n$. If $k+l \geq n$, then $0^{n-k} \oplus x(1:k) = 0^n \in S \cup \{0^n\}$ dominates x. Therefore every vertex is dominated by $S \cup \{0^n\}$.

There are $d^{n-k(t+1)} \cdot (d-1)$ vertices which begin with exactly $k(t+1) - 1$ zeros. Moreover, every vertex of $S \setminus \{0^n\}$ begins at most $n-1$ zeros. This needs that $k(t+1) < n$ or $1 \leq k \leq \left\lfloor \frac{n}{t+1} \right\rfloor$. Finally, 0^n is added to the dominating set $S \cup \{0^n\}$. Therefore the size of $S \cup \{0^n\}$ is

$$\begin{aligned}
&1 + \sum_{i=1}^{\left\lfloor \frac{n}{t+1} \right\rfloor} d^{n-i(t+1)} \cdot (d-1) \\
&= 1 + d^n(d-1) \sum_{i=1}^{\left\lfloor \frac{n}{t+1} \right\rfloor} \left(d^{-t-1}\right)^i \\
&= 1 + d^n(d-1)\left(-1 + \sum_{i=0}^{\left\lfloor \frac{n}{t+1} \right\rfloor} \left(d^{-t-1}\right)^i\right) \\
&= 1 + d^n(d-1)\left(-1 + \frac{1 - \left(d^{-t-1}\right)^{\left\lfloor \frac{n}{t+1} \right\rfloor + 1}}{1 - d^{-(t+1)}}\right) \\
&= 1 + d^n(d-1)\left(\frac{d^{-(t+1)} - \left(d^{-(t+1)}\right)^{\left\lfloor \frac{n}{t+1} \right\rfloor + 1}}{1 - d^{-(t+1)}}\right) \\
&= 1 + d^{n-t-1}(d-1)\left(\frac{1 - d^{-(t+1)\left\lfloor \frac{n}{t+1} \right\rfloor}}{1 - d^{-(t+1)}}\right).
\end{aligned}$$

□

This result gives us the following lower bound on the size of a t-dominating set in $\vec{\mathcal{B}}(d,n)$.

Theorem 4.39. *Bounds on the size of a t-dominating set in $\vec{\mathcal{B}}(d,n)$ are given by:*

$$\gamma_t(\vec{\mathcal{B}}(d,n)) \geq \begin{cases} 1 + d^{n-t-1}(d-1)\left(\frac{1-d^{-(t+1)\left\lfloor \frac{n}{t+1}\right\rfloor}}{1-d^{-(t+1)}}\right) & \underline{\text{if } n \equiv t \pmod{t+1}} \\ d^{n-t-1}(d-1)\left(\frac{1-d^{-(t+1)\left\lfloor \frac{n}{t+1}\right\rfloor}}{1-d^{-(t+1)}}\right) & \underline{\text{otherwise.}} \end{cases}$$

Proof. Suppose that the set S' is a t-dominating set in $\vec{\mathcal{B}}(d,n)$. Choose $a \in \mathcal{A}_d \backslash \{0\}$ and $i \in \mathbb{Z}$ so that $i \leq \frac{n}{t+1}$, and $w \in \mathcal{A}_d^{n-i(t+1)}$. Let $x = w \oplus a \oplus 0^{i(t+1)-1}$. We note that $B_t^-(x)$ contains the following elements.

$$B_t^-(x) = \left\{ y \mid y = w' \oplus w \oplus a \oplus 0^{i(t+1)-1-k} \text{ for some } k \in [0,t], w' \in \mathcal{A}_d^k \right\}$$

Note that for all $v \neq x$ such that $v = v' \oplus b \oplus 0^{j(t+1)-1}$ with $b \in [d-1]$, $j \leq \frac{n}{t+1}$, and $v' \in \mathcal{A}_d^{n-j(t+1)}$, we must have that $B_t^-(x) \cap B_t^-(v) = \emptyset$. Hence each of these types of strings must dominated by a different element of S', and so we must have the following lower bound on $|S'|$. Define $A = \left\{ v \mid v = v' \oplus b \oplus 0^{j(t+1)-1} \text{ with } b \in [d-1], j \leq \frac{n}{t+1}, v' \in \mathcal{A}_d^{n-j(t+1)} \right\}$.

$$\begin{aligned} |S'| &\geq |A| \\ &= \sum_{j=1}^{\lfloor \frac{n}{t+1} \rfloor} d^{n-j(t+1)} \cdot (d-1) \\ &= d^{n-t-1}(d-1)\left(\frac{1-d^{-(t+1)\left\lfloor \frac{n}{t+1}\right\rfloor}}{1-d^{-(t+1)}}\right) \end{aligned}$$

Finally, we consider the string 0^n and note that

$$B_t^-(0^n) = \{ z \mid z = z' \oplus 0^{n-s} \text{ with } z' \in \mathcal{A}_d^s, s \leq t \}.$$

When we compare $B_t^-(0^n)$ with $B_t^-(x)$, we note that since $a \neq 0$ we must have that the closest element of $B_t^-(x)$ to 0^n is x itself. Next we note that the string closest to 0^n in the set A will occur when $j = \lfloor \frac{n}{t+1} \rfloor$. This will give us the string with the most 0's packed at the right end. Finally, if $n \equiv p \bmod t+1$, then this string looks like $v' \oplus b \oplus 0^{n-p-1}$ with $v' \in \mathcal{A}_d^p$ and $b \neq 0$. If $p = t$, then we are still unable to reach 0^n, and so we must have at least one additional string in S' to cover 0^n. □

4.1.2 Undirected de Bruijn Graphs

Lemma 4.40.

$$\mathsf{dom}(\mathcal{B}(d,n)) \geq \begin{cases} \frac{d^{n-1}}{2}, & \text{if } d \text{ is even;} \\ \frac{d^{n-1}+1}{2}, & \text{if } d \text{ is odd.} \end{cases}$$

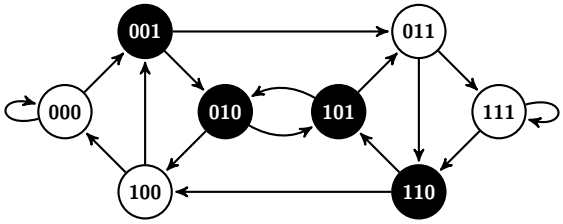


Figure 2: A directed resolving set in the graph $\vec{\mathcal{B}}(2,3)$ (black vertices).

Proof. Every vertex $v \in V(\mathcal{B}(d,n))$ can cover at most $2d+1$ vertices. Since there are d^n vertices in the graph, this provides the following lower bound.

$$\begin{aligned} \mathsf{dom}(\mathcal{B}(d,n)) &\geq \left\lceil \frac{d^n}{2d+1} \right\rceil \\ &= \begin{cases} \frac{d^{n-1}}{2}, & \text{if } d \text{ is even;} \\ \frac{d^{n-1}+1}{2}, & \text{if } d \text{ is odd.} \end{cases} \end{aligned}$$

□

4.2 Automorphisms, Resolving Sets, and Determining Sets

4.2.1 Resolving Sets

Definition 4.41. A **directed resolving set** is a set S such that for all $u, v \in V(G)$ there exist $s \in S$ so that $\vec{d}(s,u) \neq \vec{d}(s,v)$. The **directed metric dimension** is the minimum size of a directed resolving set. An example of a directed resolving set in $\vec{\mathcal{B}}(2,3)$ is given in Figure 2.

Note that this definition is not quite the same as that given in [8] (which requires that there exist $s \in S$ so that $\vec{d}(u,s) \neq \vec{d}(v,s)$). Our definition corresponds better to the definitions of domination and of identifying codes for directed graphs that are used in this paper.

Theorem 4.42. *The directed metric dimension for $\vec{\mathcal{B}}(d,n)$ is $d^{n-1}(d-1)$.*

Proof. The following shows that for each $w \in \mathcal{A}^{n-1}$ a directed resolving set for $\vec{\mathcal{B}}(d,n)$ must contain (at least) all but one of the vertices with prefix w. Suppose that $w \in \mathcal{A}^{n-1}$, and $i \neq j \in \mathcal{A}$ so that neither of $w \oplus i, w \oplus j$ is in our set S. Note that if $x, y \in V(\vec{\mathcal{B}}(d,n))$, with $x \neq y$, then the distance from x to y is completely determined by x^- (and y^+). Since neither $w \oplus i$ nor $w \oplus j$ is in S, and both have the same prefix, $\vec{d}(w \oplus i, x) = \vec{d}(w \oplus j, x)$ for all $x \in S$. Thus S is not a directed resolving set. Thus for every $w \in \mathcal{A}^{n-1}$, S must contain (at

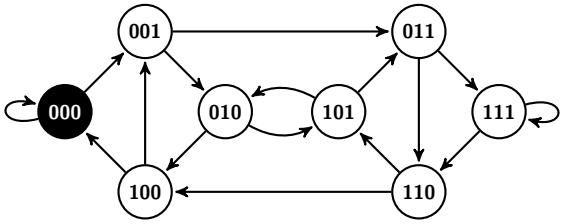


Figure 3: A minimum size determining set for $\vec{\mathcal{B}}(2,3)$ (black vertex).

least) all but one of the strings $w \oplus j$ for $j \in \mathcal{A}$. Thus $|S| \geq d^{n-1}(d-1)$. Since $\{w \oplus 0 \mid w \in \mathcal{A}^{n-1}\}$ can easily be shown to be a directed resolving set, we have the desired equality. □

The combination of Theorem 4.54 and Theorem 4.2.1 yields:

Corollary 4.43. *The directed metric dimension for $\vec{\mathcal{B}}(d,n)$ is equal to the minimum size of a t-identifying code for $\vec{\mathcal{B}}(d,n)$ if $2t \leq n$.*

4.2.2 Automorphisms and Determining Sets

In this section we will use a determining set to help us illustrate the automorphism group of $\vec{\mathcal{B}}(d,2)$, study the relationship between $\mathrm{Aut}(\vec{\mathcal{B}}(d,n-1))$ and $\mathrm{Aut}(\vec{\mathcal{B}}(d,n))$ and use the result to find the determining number for each $\vec{\mathcal{B}}(d,n)$. First let's recall some definitions.

Definition 4.44. An **automorphism** of a graph G is a permutation π of the vertex set such that for all pairs of vertices $u, v \in V(G)$, uv is an edge between u and v if and only if $\pi(u)\pi(v)$ is an edge between $\pi(u)$ and $\pi(v)$. An **automorphism** of a directed graph G is a permutation π of the vertex set such that for all pairs of vertices $u, v \in V(G)$, uv is an edge from u to v if and only if $\pi(u)\pi(v)$ is an edge from $\pi(u)$ to $\pi(v)$. One automorphism in the binary (directed or undirected) de Bruijn graph is a map that sends each string to its complement.

Definition 4.45. [6] A **determining set** for G is a set S of vertices of G with the property that the only automorphism that fixes S pointwise is the trivial automorphism. The **determining number** of G, denoted $\mathrm{Det}(G)$ is the minimum size of a determining set for G. See Figure 3 for an example.

Note that an alternate definition for a determining set is a set S with the property that whenever $f, g \in \mathrm{Aut}(G)$ so that $f(s) = g(s)$ for all $s \in S$, then $f(v) = g(v)$ for all $v \in V(G)$. That is, every automorphism is completely determined by its action on a determining set.

Notice that since for both directed resolving sets and for identifying codes, since each vertex in a graph is uniquely identified by its relationship to the subset

by properties preserved by automorphisms, the subset it also a determining set. Thus every directed resolving set and every identifying code is a determining set. However, though domination is preserved by automorphisms, vertices are not necessarily uniquely identifiable by their relationship to a dominating set. Thus a dominating set is not necessarily a determining set. However, the relationships above mean that the size of a minimum determining set must be at most the size of a minimum identifying code or the directed metric dimension. For de Bruijn graphs we have shown that the latter numbers are rather large. Does this mean that the determining number is also large. We will see in Corollary 4.49 that the answer for directed de Bruijn graphs is a resounding 'No'.

Lemma 4.46. *$S = \{00, 11, 22, 33, \ldots, (d-1)(d-1)\}$ is a determining set for $\vec{\mathcal{B}}(d, 2)$.*

Proof. Suppose that $\sigma \in \mathrm{Aut}(\vec{\mathcal{B}}(d, 2))$ fixes S pointwise. That is, $\sigma(ii) = ii$ for all $i \in \mathcal{A}$. Choose $ij \neq rs \in V(\vec{\mathcal{B}}(d, 2))$. Then either $i \neq r$ or $j \neq s$ (or both). If $i \neq r$ then $\vec{d}(ii, rs) = 2$ which is distinct from $\vec{d}(ii, ij) = 1$. Since an automorphism of a directed graph must preserve directed distance, $\sigma(ij) \neq rs$ if $i \neq r$. If $j \neq s$, then $\vec{d}(rs, jj) = 2$ which is distinct from $\vec{d}(ij, jj) = 1$. Thus, again using that σ preserves directed distance, $\sigma(ij) \neq rs$ if $j \neq s$. Thus, $\sigma(ij) = ij$ for all $ij \in V(\vec{\mathcal{B}}(d, 2))$ and therefore σ is the identity map and S is a determining set. □

Note that we are using directed distances both from and to elements of the set S. Thus S does not fit the definition of a directed resolving set for $\vec{\mathcal{B}}(d, 2)$ (by [8], this would require that each vertex $v \in V(G)$ be distinguished by it directed distance to the vertices of the resolving set). However directed distances both to and from a set can be used in determining automorphisms of a directed graph.

Lemma 4.47. $\mathrm{Aut}(\vec{\mathcal{B}}(d, 2)) \cong \mathrm{Sym}(\mathcal{A}_d)$.

Proof. Let $\sigma \in \mathrm{Sym}(\mathcal{A}_d)$. Define φ_σ on $V(\vec{\mathcal{B}}(d, n))$ by applying σ to each vertex coordinate-wise. That is $\varphi_\sigma(ab) = \sigma(a)\sigma(b)$. It is easy to show that φ_σ preserves directed edges and thus is an automorphism. Further, distinct permutations in $\mathrm{Sym}(\mathcal{A}_d)$ produce distinct automorphisms since they act differently on the vertices of the determining set S defined above. Thus we have an injection $\mathrm{Sym}(\mathcal{A}_d) \hookrightarrow \mathrm{Aut}(\vec{\mathcal{B}}(d, 2))$.

Since the vertices of S are precisely the vertices with loops, every automorphism of $\vec{\mathcal{B}}(d, 2)$, must preserve S setwise. This provides the necessary injection from $\mathrm{Aut}(\vec{\mathcal{B}}(d, 2)) \hookrightarrow \mathrm{Sym}(\mathcal{A})$. Thus, $\mathrm{Aut}(\vec{\mathcal{B}}(d, 2)) \cong \mathrm{Sym}(\mathcal{A}_d)$. □

Note that we can consider the automorphisms of $\vec{\mathcal{B}}(d, 2)$ as permutations of the loops, but we can simultaneously consider them as permutations of the symbols in the alphabet $\mathcal{A}_d$. It can be useful to view the automorphisms in these two different ways.

Note that as shown in [1], $\vec{\mathcal{B}}(d, n)$ can be built inductively from $\vec{\mathcal{B}}(d, n-1)$ in the following way. The vertex $x_1 \ldots x_n \in V(\vec{\mathcal{B}}(d, n))$ corresponds to the directed

edge from $x_1 \dots x_{n-1}$ to $x_2 \dots x_n$ in $\vec{\mathcal{B}}(d, n-1)$. The directed edge $\vec{\mathcal{B}}(d, n)$ from $x_1 \dots x_n \to x_2 \dots x_n x_{n+1}$ corresponds to the directed 2-path $x_1 \dots x_{n-1} \to x_2 \dots x_n \to x_3 \dots x_{n+1}$ in $\vec{\mathcal{B}}(d, n-1)$. That is, $\vec{\mathcal{B}}(d, n)$ is the directed line graph of the directed graph $\vec{\mathcal{B}}(d, n-1)$. Thus, by [13] (Chapter 27, Section 1.1), $\mathrm{Aut}(\vec{\mathcal{B}}(d, n-1)) = \mathrm{Aut}(\vec{\mathcal{B}}(d, n)) \cong \mathrm{Sym}(\mathcal{A})$. In the following paragraphs we see detail this correspondence.

Suppose that $\varphi \in \mathrm{Aut}(\vec{\mathcal{B}}(d, n))$. Since φ preserves directed edges, we know that both $\varphi(x_1 \dots x_n) = a_1 \dots a_n$ and $\varphi(x_2 \dots x_{n+1}) = b_1 \dots b_n$ if and only if $a_2 = b_1, \dots, a_n = b_{n-1}$. Thus if $\varphi(x_1 \dots x_{n-1} x_n) = a_1 \dots a_{n-1} a_n$ then for every $b \in \mathcal{A}$, $\varphi(x_1 \dots x_{n-1} z) = a_1 \dots a_{n-1} c$ for some $c \in \mathcal{A}$. In particular, this allow us to define an automorphism $\varphi' \in \mathrm{Aut}(\vec{\mathcal{B}}(d, n-1))$ corresponding to $\varphi \in \mathrm{Aut}(\vec{\mathcal{B}}(d, n))$. Define φ' by $\varphi'(x_1 \dots x_{n-1}) = a_1 \dots a_{n-1}$ where $\varphi(x_1 \dots x_n) = a_1 \dots a_{n-1}$. By the preceding discussion, φ' is well-defined. It is also clearly a bijection on vertices of $\vec{\mathcal{B}}(d, n-1)$. Consider $x_1 \dots x_{n-1}$ and $x_2 \dots x_{n-1} x_n$, the initial and terminal vertices of a directed edge in $\vec{\mathcal{B}}(d, n-1)$. Since φ preserves directed edges if $\varphi(x_1 \dots x_n) = a_1 \dots a_{n-1} a_n$ then for any $z \in \mathcal{A}$, $\varphi(x_2 \dots x_n z) = a_2 \dots a_n w$ for some $w \in \mathcal{A}$. By definition of φ',$\varphi'(x_1, \dots x_{n-1}) = a_1 \dots a_{n-1}$ and $\varphi'(x_2 \dots x_n) = a_2 \dots a_n$. Thus φ' preserves the directed edge. Thus we get $\mathrm{Aut}(\vec{\mathcal{B}}(d, n)) \hookrightarrow \mathrm{Aut}(\vec{\mathcal{B}}(d, n-1))$.

In the other direction, suppose we are given $\varphi' \in \mathrm{Aut}(\vec{\mathcal{B}}(d, n-1))$. Since φ' preserves directed edges, and directed edges of $\vec{\mathcal{B}}(d, n-1)$ are precisely the vertices of $\vec{\mathcal{B}}(d, n)$, φ' defines a map on vertices of $\vec{\mathcal{B}}(d, n)$. That is, (with some abuse of notation)

$$\begin{aligned} \varphi(x_1 \dots x_n) &= \varphi(x_1 \dots x_{n-1} \to x_2 \dots x_n) \\ &= \varphi'(x_1 \dots x_{n-1} \to x_2 \dots x_n) \\ &= \varphi'(x_1 \dots x_{n-1}) \to \varphi'(x_2 \dots x_n). \end{aligned}$$

Thus, given $\varphi'(x_1 \dots x_{n-1}) = a_1 \dots a_{n-1}$ then $\varphi'(x_2 \dots x_n) = a_2 \dots a_n$ for some $a_n \in \mathcal{A}$ and we define $\varphi(x_1 \dots x_n) = a_1 \dots a_n$. Further, since φ' preserves directed 2-paths, φ preserves directed edges. Thus we get

$$\mathrm{Aut}(\vec{\mathcal{B}}(d, n-1)) \hookrightarrow \mathrm{Aut}(\vec{\mathcal{B}}(d, n)).$$

Since the automorphisms of $\vec{\mathcal{B}}(d, 2)$ are permutations of the loops, and of the symbols of $\mathcal{A}$, by induction, so are the automorphisms of $\vec{\mathcal{B}}(d, n)$ for all n. Thus we have proved the following.

Theorem 4.48. $\mathrm{Aut}(\vec{\mathcal{B}}(d, n)) \cong \mathrm{Sym}(\mathcal{A}_d)$ *for all* $n \geq 2$.

Corollary 4.49. $\mathrm{Det}(\vec{\mathcal{B}}(d, n)) = \left\lceil \frac{d-1}{n} \right\rceil$.

Proof. Let S be a minimum set of vertices in which each letter of $\mathcal{A}_{d-1}$ occurs at least once. It is easy to see that $|S| = \left\lceil \frac{d-1}{n} \right\rceil$. Any permutation of $\mathcal{A}_d$ that acts nontrivially on any letter of $\mathcal{A}_d$ must act non-trivially on any string containing

that letter. Thus if $\sigma \in \mathrm{PtStab}(S)$, then σ must fix every letter contained in any string in S. Thus σ fixes $0, 1, \ldots, d-1$ and therefore also d. We can conclude that σ is the identity in both $\mathrm{Sym}(\mathcal{A}_d)$ and in $\mathrm{Aut}(\vec{\mathcal{B}}(d,n)$ and therefore S is a determining set. Thus $\mathrm{Det}(\vec{\mathcal{B}}(d,n)) \leq \lceil \frac{d-1}{n} \rceil$.

Further if $|S| < \lceil \frac{d-1}{n} \rceil$ then fewer than $d-1$ letters of $\mathcal{A}_d$ are used in strings in S. If $a, b \in A_d$ are not represented in S, then the transposition $(a\ b)$ in $\mathrm{Sym}(A_d)$ is a non-trivial automorphism of $\vec{\mathcal{B}}(d,n)$ that fixes S pointwise. Thus S is not a determining set. □

Thus for directed de Bruijn graphs, the determining number and the directed metric dimension can be vastly different in size.

4.3 Identifying Codes

We have a few general results that hold true for any graph. The first result will be used frequently in proving the existence of t-identifying codes in a graph.

Definition 4.50. Two vertices $u, v \in V(G)$ are called **t-twins** whenever $B_t^-(u) = B_t^-(v)$. If the graph has no t-twins, then G is called **t-twin-free**.

Theorem 4.51 ([7])**.** *For a given graph G and integer t, G has a t-identifying code if and only if it is t-twin-free.*

Next, we prove some inductive relationships that exist with identifying codes that are useful.

Theorem 4.52. *If $\mathcal{G}$ is t-identifiable, then it is also $(t-1)$-identifiable.*

We will prove this result using the converse of the following lemma.

Lemma 4.53. *Suppose that $\{x, y\}$ are t-twins in $\mathcal{G}$. Then we must also have that $\{x, y\}$ are $(t+1)$-twins.*

Proof. First, we note that if $\{x, y\}$ are t-twins, then $d(x, y) \leq t$, so x and y must be in the same component C of $\mathcal{G}$. If $B_t(x) = B_t(y) = C$, then clearly x and y are $(t+1)$-twins, as there are no vertices z such that $d(x, z) = t+1$ or $d(y, z) = t+1$. However, if $B_t(x) \subsetneq C$, then there exists some $\beta \in C$ such that $d(x, \beta) = t+1$. We know that β must have some neighbor α such that $d(x, \alpha) = t$. Note that since $d(x, \beta) > t$ and $B_t(x) = B_t(y)$, we also must have $d(y, \beta) > t$ and $d(y, \alpha) \leq t$. Since $d(\alpha, \beta) = 1$, this implies that $d(y, \beta) = t+1$ and hence we have $\beta \in B_{t+1}(y)$. Since β was arbitrary, this implies that $B_{t+1}(x) \subseteq B_{t+1}(y)$. The same argument follows with x and y reversed to show that $B_{t+1}(y) \subseteq B_{t+1}(x)$, and hence $B_{t+1}(x) = B_{t+1}(y)$ and we have that $\{x, y\}$ are $(t+1)$-twins in $\mathcal{G}$. □

We note that in the literature others have considered various classes of graphs, such as interval graphs and permutations graphs [12], however the vast majority of these results do not pertain to de Bruijn networks in particular.

While these classes of graphs do not contain the class of de Bruijn graphs, the class of undirected line graphs is considered in [11]. Due to the recursive nature of the de Bruijn graphs, it is clear that they are indeed line graphs, and these results will be discussed further in Section 4.3.2.

4.3.1 Directed de Bruijn Graphs

We begin by considering directed de Bruijn graphs. Similar to the results found for dominating sets, the directed graphs have much simpler and obvious identifying code constructions.

Theorem 4.54. *If $\vec{\mathcal{B}}(d,n)$ is a t-identifiable graph, then the size of any t-identifying code is at least $d^{n-1}(d-1)$.*

Proof. Choose $t \leq n$ and $a \neq b$ in $\mathcal{A}$. Suppose that for some $w \in \mathcal{A}^{n-1}$, neither $x = w \oplus a$ nor $y = w \oplus b$ is a set S. Since x and y share a prefix, by Lemma 3.33, $B_t^-(x) \setminus \{x\} = B_t^-(y) \setminus \{y\}$. Since neither x nor y is in S, $\mathsf{ID}_S(x) = \mathsf{ID}_S(y)$. Thus S is not a t-identifying code. Thus for each $w \in \mathcal{A}^{n-1}$, a t-identifying code must contain, at least, all but one of $w \oplus a$ for $a \in \mathcal{A}$. Thus a t-identifying code for $\vec{\mathcal{B}}(d,n)$ must have size at least $d^{n-1}(d-1)$. □

Note that the result above is independent of the radius t. An interesting consequence of this is the fact that increasing the radius of our identifying code does not produce any decrease in the size of a minimum identifying code. For example, consider the potential application of identifying codes in sensor networks. One might think that by increasing the sensing power (which corresponds to the radius of the identifying code) we would be able to place fewer sensors and thus incur a savings overall. However, Theorem 4.54 implies that providing more powerful (and thus more expensive) sensors does not allow us to place fewer sensors. Thus we should use sensors that have sensing distance equivalent to radius one. In fact, in the case of 2-identifying codes in $\vec{\mathcal{B}}(2,3)$, we actually require an extra vertex for a minimum size of seven!

The remainder of this section is organized as follows. We first provide a construction of an optimal t-identifying code for $\vec{\mathcal{B}}(d,n)$ with $t \geq 2$, and $n \geq 2t$ in Theorems 4.55 and 4.56. Following the proof of this result, we highlight some variations that provide identifying codes for several other instances. Finally, we highlight an alternative construction for 1-identifying codes for all $\vec{\mathcal{B}}(d,n)$ when $d \neq 2$ and n is odd.

Theorem 4.55. *Suppose that $n \geq 5, d \geq 2, t \geq 2$, and $n \geq 2t$. Then the following set S is an optimal t-identifying code of size $d^{n-1}(d-1)$ in $\vec{\mathcal{B}}(d,n)$.*

$$\begin{aligned} S \;=\; & \Big\{ x \in \mathcal{A}_d^n \mid \text{for some } m \text{ and } \ell \leq t, x^{(t,m)}(t+1-\ell : n-1) \text{ is } \ell\text{-periodic,} \\ & \qquad \text{but } x^{(t,m)}(t+1-\ell : n-1) \oplus (x_n + m) \text{ is not.} \Big\} \\ & \qquad\qquad \cup \\ & \Big\{ x \in \mathcal{A}_d^n \mid x_t \neq x_n \text{ and } x^{(t,m)}(t+1-\ell : n-1) \\ & \qquad \text{is not } \ell\text{-periodic for any } m \text{ and } \ell \leq t \} \end{aligned}$$

Proof. First, let us note that for all $x \in \mathcal{A}_d^n$, if $x^{(t,m)}(t+1-\ell : n)$ has period ℓ for any m and $\ell \leq t$, then $x \notin S$.

Let x be given, and define x_i to be the ith coordinate of x. We also define the following string x_{ij}.

$$x_{ij}(k) = \begin{cases} i, & \text{if } k = t; \\ j, & \text{if } k = n; \\ x(k), & \text{otherwise.} \end{cases}$$

In other words, x_{ij} is the string $x_1 \dots x_{t-1} i x_{t+1} \dots x_{n-1} j$.

Next, we note that $x_{ij}(t+1-\ell : n-1) = x^{(t,i-x_t)}(t+1-\ell : n-1)$, which implies by Lemma 3.30 that $x_{ij}(t+1-\ell : n-1)$ is ℓ-periodic for at most one $\ell \leq t = \frac{n}{2}$ and for at most one $i \in \mathcal{A}_d$.

Next, suppose that $x_{ab}(t+1-\ell : n)$ is ℓ-periodic and consider x_{ij}. We note that $x_{ij}^{(t,a-i)}(t+1-\ell : n-1) = x_{ab}(t+1-\ell : n-1)$, which is ℓ-periodic. Hence x_{ij} is a member of S if and only if $x_{ij}^{(t,a-i)}(t_1 - \ell : n-1) \oplus (j+a-i)$ is not ℓ-periodic. In other words, if and only if $i - j \not\equiv a - b \bmod d$. On the other hand, if $x_{ij}(t+1-\ell : n-1)$ is not ℓ-periodic for any ℓ and $i \in \mathcal{A}_d$, then $x_{ij} \in S$ if and only if $i \neq j$. Hence, there is exactly one $j \in \mathcal{A}_d$ such that $x_{aj} \notin S$, and similarly there is exactly one $i \in \mathcal{A}_d$ such that $x_{ib} \notin S$. These pairings tell us that d^{n-1} strings are not in S, leaving the cardinality of S at $d^n - d^{n-1}$, or $d^{n-1}(d-1)$.

Now that we have established the cardinality of S, we must show that no two nodes have the same identifying sets. Let $x, y \in \vec{\mathcal{B}}(d,n)$, and consider their identifying sets, called $I(x)$ and $I(y)$, respectively. Let k be the smallest index such that $x_k \neq y_k$.

$k = 1$: Without loss of generality, we may assume that $x_1 = 0$ and $y_1 = 1$. Observe that we have the following strings contained in the identifying sets.

$$\begin{array}{rcccl} x' & = & 0^{t-1} \oplus 0 \oplus x(1:n-t) & \in & B_t^-(x), \\ x'' & = & 0^{t-1} \oplus 1 \oplus x(1:n-t) & \in & B_t^-(x), \\ y' & = & 1^{t-1} \oplus 1 \oplus y(1:n-t) & \in & B_t^-(y), \text{ and} \\ y'' & = & 1^{t-1} \oplus 0 \oplus y(1:n-t) & \in & B_t^-(y). \end{array}$$

Note that $x' \notin B_t^-(y)$ and $y' \notin B_t^-(x)$, since $B_t^-(x)$ does not contain any vertices beginning with 1^{t+1} and $B_t^-(y)$ does not contain any vertices beginning with 0^{t+1}. Next, we notice that at least one $\{x', x''\}$ is in S. To see this, we note that $x'' = x'^{(t,1)}$. By the same point, we must have that at least one of y', y'' is a member of S.

Next, we note that if at least one of x', y' is a member of S, we can use that string to separate y. Otherwise, if either $x'' \notin B_t^-(y) \cap S$ or $y'' \notin B_t^-(x) \cap S$, we can separate x and y with the given string. As a last resort, we consider the case in which we have $x'' \in B_t^-(y) \cap S$ and

$y'' \in B_t^-(x) \cap S$. Then we must have:

$$\begin{aligned} x'' &= 0^{t-1} \oplus y(1 : n-t+1), \\ y'' &= 1^{t-1} \oplus x(1 : n-t+1), \end{aligned}$$

as x''_t is the only 1 in $x''(1 : t+1)$ and y''_t is the only 0 in $y''(1 : t+1)$. From this, we get the following string equalities:

$$\begin{aligned} x(1 : n-t) &= y(2 : n-t+1), \\ y(1 : n-t) &= x(2 : n-t+1). \end{aligned}$$

From these string equalities, we see that $x_i = y_{i+1} = x_{i+2}$ and $y_i = x_{i+1} = y_{i+2}$ for all $i = 1, 2, ..., n-t-1$, and so $x(1 : n-t)$ and $y(1 : n-t)$ are both 2-periodic (since $x_1 \neq y_1$ we cannot have 1-periodic). Hence, $x''(t-1 : n) = 0 \oplus 1 \oplus x(1 : n-t)$ is also 2-periodic (recall that $x_1 = 0$ and $y_1 = 1$), so $x'' \notin S$. Hence we must have $x' \in S$, and so we may use x' to separate x and y.

$2 \leq k \leq n-t$: We know that there must exist some s such that $x_1 \dots x_{s-1} = y_1 \dots y_{s-1}$, and these substrings are constant. Without loss of generality we may assume that $x_1 \dots x_{s-1} = 0^{s-1} = y_1 \dots y_{s-1}$ and $x_s = 1$, and so $2 \leq s \leq k$. Define the following strings.

$$\begin{aligned} y' &= 1^t \oplus y(1 : n-t) \text{ and} \\ y'' &= 1^{t-1} \oplus 0 \oplus y(1 : n-t) \end{aligned}$$

As we saw in the previous case, we have that $\{y', y''\} \subseteq B_t^-(y)$ and $\{y', y''\} \cap S \neq \emptyset$. Now consider an arbitrary vertex $v \in B_t^-(x)$. Since $x_{s-1}x_s = 01$ and $2 \leq s \leq k \leq n-t$, we know that $v(i-1 : i) = 01$ for some $i \in [s, s+t]$.

Additionally, we consider y' and $i \in [2, s+t-1]$. For $i \leq t$, we know that $y'(i-1 : i) = 11$, and for $i = t+1$, we have that $y'(i-1 : i) = 10$, and finally for $t+2 \leq i \leq s+t-1$ we have $y'(i-1 : i) = 00$. Similarly, for $i \in [2, s+t-1]$, we must have $y''(i-1 : i) \in \{00, 10, 11\}$. This implies that $y'(1 : s+t-1)$ and $y''(1 : s+t-1)$ do not contain the substring 01, and so if y' or y'' is a member of $B_t^-(x)$, we must have either $d(y', x) = t$ or $d(y'', x) = t$, respectively. Hence we must have $y'_{t+i} = x_i$ or $y''_{t+i} = x_i$ for $i \in [1, n-t]$, and therefore that $x_k = y'_{t+k} = y''_{t+k} = y_k$, which is a contradiction. Thus neither y' nor y'' can be a member of $B_t^-(x)$, and hence both strings separate x and y.

$k > n-t$: Since we must have $x_1 = y_1$, we may assume without loss of generality that $x_1 = y_1 \neq 0$. Define the following strings.

$$\begin{aligned} u &= 0^{n-k} \oplus x(1 : k) \text{ and} \\ v &= 0^{n-k} \oplus y(1 : k) \end{aligned}$$

Clearly we have $u \in B_t^-(x)$ and $v \in B_t^-(y)$. Additionally, we have $u(1 : n-1) = v(1 : n-1)$ and $u_n = x_k \neq y_k = v_n$. Note that this implies that for $a = x_{t-n+k}$ we have $u = u_{a,x_k}$ and $v = u_{a,y_k}$ with $x_k \neq y_k$. By our argument at the very beginning of the proof, at most one of these can lie outside of S, and so we must have $\{u, v\} \cap S \neq \emptyset$.

We now have two cases. First, if both $u \notin B_t^-(y)$ and $v \notin B_t^-(x)$, then any string from $\{u, v\} \cap S$ separates x and y, and we are done. Otherwise, assume without loss of generality that $u \in B_t^-(y)$. Then we must have $u = w \oplus y(1 : p)$ for at least one $p \in [n-t, n]$. Take p to be the largest such p possible. Since $y_1 \neq 0 = u_i$ for all $i \in [1, n-k]$, we must have $p \leq k$. Additionally, if $p = k$ then we have $y(1 : k) = y(1 : p) = x(1 : k)$, which is a contradiction since $y_k \neq x_k$. This implies that we must have $p < k$. Hence we must have the following string of equalities:

$$\begin{aligned} 0^{n-k} \oplus x(1 : k) &= u \\ &= w \oplus y(1 : p) \\ &= w \oplus x(1 : p). \end{aligned}$$

Thus we have $x(k-p+1 : k) = x(1 : p)$, or $x_i = x_{k-p+i}$ for $i = 1, 2, \ldots, p$. Additionally we note the following equalities hold:

$$\begin{aligned} 2(k-p) &= 2k - 2p \\ &\leq 2k - 2(n-t) \\ &\leq k + 2t - n \\ &\leq k. \end{aligned}$$

Hence since $x_i = x_{k-p+i}$ for $i \in [1, p]$ and $2(k-p) \leq k$, we know that $x(1 : k)$ has period $(k-p)$. In fact, since we chose p to be maximum, $x(1 : k)$ is $(k-p)$-periodic.

Next, we show that $u(t+1-(k-p) : n)$ is also $(k-p)$-periodic. First, we note the following inequalities:

$$\begin{aligned} n - (t+1-(k-p)) + 1 &= n - t + k - p \\ &\geq 2(k-p). \end{aligned}$$

The last line comes from the facts: $n - t \geq t \geq n - p \geq k - p$. Hence our string length is at least $2(k-p)$, and from our previous paragraphs, so long as $u(t+1-(k-p) : n)$ is contained in $u(n-k+1 : n) = x(1 : k)$, we know that it must have period $(k-p)$. For this we note that

$$t + 1 - (k-p) \geq (n-p) + 1 - (k-p) = n - k + 1,$$

and so $u(t+1-(k-p) : n)$ indeed has period $(k-p)$, and thus $u \notin S$, except if $(n-1) - (t+1-(k-p)) + 1 < 2(k-p)$. In this case we must have $k - p = t$, $k = n = 2t$, and $p = t$.

If $u^{(t,m)}(t+1-\ell : n-1)$ is ℓ-periodic for some $\ell < k-p = t$ and m, then by Lemma 3.31 we must have that $u^{(t,m)}(t+1-\ell : n-1) \oplus (u_n + m)$ is also ℓ-periodic, and hence $u \notin S$. On the other hand, if $u^{(t,m)}(t+1-\ell : n-1)$ is not ℓ-periodic for any $\ell < t$ and m, then we note that $u_n = u_{2t} = u_t$, so again $u \notin S$. Therefore in all cases we have $u \notin S$, so we must have $v \in S$.

All that remains is to show that $v \notin B_t^-(x)$. We note that if $v \in B_t^-(x)$, then $y(1:k)$ is ℓ-periodic for some $\ell \leq k+t-n$ (using the same argument as we used to show that $x(1:k)$ was $(k-p)$-periodic). Since $x(1:k) = (y(1:k))^{(k,m)}$ for some m, by Lemma 3.29 it is not possible that both $x(1:k)$ and $y(1:k)$ are periodic. Hence we must have $v \notin B_t^-(x)$, and so we may use v to separate x and y.

□

When we combine the previous theorem with the following result, we have a complete set of constructions for optimal t-identifying codes in $\vec{\mathcal{B}}(d,n)$ with $t \geq 2$ and $n \geq 2t$.

Theorem 4.56. *Let $n > 3$ and $S = \mathcal{A}_d^n \setminus \{x_1 a x_3 x_4 \dots x_{n-1} a \mid a \in \mathcal{A}_d\}$. If n is even, then S is a 2-identifying code for $\vec{\mathcal{B}}(d,n)$. If n is odd, then $S' = (S \cup \{(ab)^{\frac{n-1}{2}} b \mid \ a \neq b \in \mathcal{A}_2\}) \setminus \{(ab)^{\frac{n-1}{2}} a \mid \ a \neq b \in \mathcal{A}_2\}$ is a 2-identifying code for $\vec{\mathcal{B}}(d,n)$. In both these cases, the 2-identifying code is of optimal size $d^{n-1}(d-1)$.*

Proof. Consider an arbitrary string $x = x_1 x_2 x_3 \dots x_n \in \mathcal{A}_d^n$, and define the set $T = \mathsf{ID}_S(x)$. We'll consider the contents of T in four cases based on the equality of x_1, x_{n-1} and of x_2, x_n. First let $C = \{a x^{--} \mid a \in \mathcal{A} \setminus \{x_{n-2}\}\}$.

Case 1. If $x_2 = x_n$ and $x_1 = x_{n-1}$, then $T = \mathcal{A} \oplus C$. Thus $|T| = d^2 - d$.

Case 2. If $x_2 \neq x_n$ and $x_1 = x_{n-1}$, then $T = (\mathcal{A} \oplus C) \cup \{x\}$. If $x \in \mathcal{A} \oplus C$ then $x^+ = a x^{--}$ for some $a \in \mathcal{A} \setminus \{x_{n-2}\}$. In this case,we have $x_2 x_3 \cdots x_n = a x_1 x_2 \cdots x_{n-2}$. This implies that we have $x_1 = x_3 = x_5 = \cdots$, and also that $x_2 = x_4 = x_6 = \cdots$. Since this case requires that $x_1 = x_{n-1}$, we must have that either n is even or that $x_1 = x_2 = x_3 = \cdots = x_n$. In either case, this contradicts our assumption that $x_2 \neq x_n$. Thus $x \notin \mathcal{A} \oplus C$, and we conclude that $|T| = d^2 - d + 1$.

Case 3. If $x_2 = x_n$ and $x_1 \neq x_{n-1}$, then $T = \mathcal{A} \oplus \{C \cup \{x^-\}\}$. If $x^- = a x^{--}$ for some $a \in \mathcal{A} \setminus \{x_{n-2}\}$, then $a x_1 x_2 \cdots x_{n-2} = x_1 x_2 \cdots x_{n-1}$. This implies that we have $x_1 = x_2 = x_3 = \cdots = x_{n-2} = x_{n-1}$. This contradicts our assumption that $x_1 \neq x_{n-1}$. Thus $x^- \neq a x^{--}$ for any $a \in \mathcal{A} \setminus \{x_{n-2}\}$, and we conclude that $|T| = d^2$.

Case 4. If $x_2 \neq x_n$ and $x_1 \neq x_{n-1}$, then $T = (\mathcal{A} \oplus \{C \cup \{x^-\}\}) \cup \{x\}$. As in Case 3, since $x_1 \neq x_{n-1}$, $\mathcal{A} \oplus \{C \cup \{x^-\}\}$ contains d^2 distinct elements. Let us consider whether $x \in \mathcal{A} \oplus \{C \cup \{x^-\}\}$. If not, then $|T| = d^2 + 1$. There are two cases to consider.

a. If $x \in \mathcal{A} \oplus C$, then $x^+ = x^-$. In this case, we must have that $x_2x_3x_4 \cdots x_n = x_1x_2 \cdots x_{n-1}$, which implies that we have the following chain of equalities: $x_1 = x_2 = x_3 = \cdots = x_{n-1} = x_n$. This contradicts the assumptions that $x_2 \neq x_n$ and $x_1 \neq x_{n-1}$. Thus, this case does not occur.

b. If $x \in \mathcal{A} \oplus \{x^-\}$, then $x^+ = ax^{--}$ for some $a \in \mathcal{A}$. Then $x_2x_3 \cdots x_n = ax_1x_2 \cdots x_{n-2}$. This implies that $x_1 = x_3 = x_5 = \cdots$, and also that $x_2 = x_4 = x_6 = \cdots$. If n is even, this contradicts our assumptions that $x_2 \neq x_n$ and $x_1 \neq x_{n-1}$. Thus for even n, this case does not occur. For n odd, this case only occurs if $x \in \{(ab)^{\frac{n-1}{2}}a\}$.

Thus, if n is even, or n is odd and $x \notin \{(ab)^{\frac{n-1}{2}}a\}$, we can see that $T = \mathsf{ID}_S(x)$ completely determines the string x. In particular, given T we can decide which case we are in based on $|T|$. We can then determine $x_1, \ldots, x_n$ based on the content of T. Thus in these cases S is an identifying code.

However, if n is odd, and $x \in \{(ab)^{\frac{n-1}{2}}a\}$ we must change S to get an identifying code. Note that $B_2^-((ab)^{\frac{n-1}{2}}a) \cup \{(ab)^{\frac{n-1}{2}}b\} = B_2^-((ab)^{\frac{n-1}{2}}b)$. Since our set S contains vertices of the form $(ab)^{\frac{n-1}{2}}a$ but not $(ab)^{\frac{n-1}{2}}b$, these two types of vertices must have identical identifying sets with respect to S. Thus by adding the vertices in $\{(ab)^{\frac{n-1}{2}}b\}$, we are able to create distinct identifying sets with respect to $S \cup \{(ab)^{\frac{n-1}{2}}b\}$. However, we note that we now have the vertices of $\{(ab)^{\frac{n-1}{2}}b\}$ and $\{(ab)^{\frac{n-1}{2}}a\}$ in our identifying code, but that $B_2^+((ab)^{\frac{n-1}{2}}a) \cup \{b(ba)^{\frac{n-1}{2}}\} = B_2^+(b(ba)^{\frac{n-1}{2}})$. This implies that the inclusion of both $(ab)^{\frac{n-1}{2}}b$ and $(ab)^{\frac{n-1}{2}}a$ in our identifying code is only necessary if they are required to identify vertex $(ab)^{\frac{n-1}{2}}a$ from vertex $b(ba)^{\frac{n-1}{2}}$. So, as long as we can identify $(ab)^{\frac{n-1}{2}}a$ differently from $b(ba)^{\frac{n-1}{2}}$ without using $b(ba)^{\frac{n-1}{2}}$, we need only include $(ab)^{\frac{n-1}{2}}b$ and not $(ab)^{\frac{n-1}{2}}a$ in our identifying code. Since these two vertices have disjoint in-balls of radius 2 for $n > 3$, they must have distinct 2-identifying sets. Thus S' is a 2-identifying code in this case. □

Finally, we provide additional constructions of identifying codes. Theorems 4.57 and 4.58 have proofs very similar to that of Theorem 4.55, so we omit them here.

Theorem 4.57. *Assume that $d \geq 2$, and $n \geq 3$. Then the following subset S is an optimal 1-identifying code of size $d^{n-1}(d-1)$ in $\vec{\mathcal{B}}(d,n)$.*

$$\begin{aligned} S \;=\; & \Big\{x \in \mathcal{A}_d^n \mid \textit{for some } m \textit{ and } \ell \in \{1,2\}, x^{(1,m)}(1:n-1) \textit{ is } \ell\textit{-periodic or} \\ & \qquad \textit{almost } \ell\textit{-periodic, but } x^{(1,m)}(1:n-1) \oplus (x_n + m) \textit{ is not.}\Big\} \\ & \qquad\qquad \cup \\ & \Big\{x \in \mathcal{A}_d^n \mid x_1 \neq x_n \textit{ and } x^{(1,m)}(1:n-1) \\ & \qquad \textit{is not } \ell\textit{-periodic for any } m \textit{ and } \ell \in \{1,2\}\Big\} \end{aligned}$$

Theorem 4.58. *Assume that $d \geq 2$. Then the following subset S is a t-identifying code of size $d^{n-1}(d-1)+d^t$ in the directed de Bruijn graph $\vec{\mathcal{B}}(d,n)$, if $n = 2t-1 \geq 5$.*

$$\begin{aligned} S = &\{x \in \mathcal{A}_d^n \mid \text{for some } m \text{ and } l < t-1\text{: } x^{(t,m)}(t+1-l : n-1) \text{ is } l\text{-periodic,} \\ &\qquad \text{but } x^{(t,m)}(t+1-l : n-1) \oplus (x_n + m) \text{ is not.}\} \\ &\cup \\ &\Big\{x \in \mathcal{A}_d^n \mid x_t \neq x_n \text{ and } x^{(t,m)}(t+1-l : n-1) \text{ is not } l\text{-periodic} \\ &\qquad \text{for any } m \text{ and } l < t-1.\} \\ &\cup \\ &\{x \in \mathcal{A}_d^n \mid x^{(t,m)}(1 : n-1) \text{ is almost } t\text{-periodic, for some } m.\} \end{aligned}$$

We note that the construction in Theorem 4.58 is not optimal. To find an optimal t-identifying code when $n = 2t-1$ is an open problem to be considered in the future. For the cases when $n < 2t-1$, we have the following theorem.

Theorem 4.59. *There is no t-identifying code in the directed de Bruijn graph $\vec{\mathcal{B}}(d,n)$ when $n \leq 2t-2$.*

Proof. Let $u = 0^{n-t} \oplus 1 \oplus 0^{t-2} \oplus 1$ and $v = 0^{n-t} \oplus 1 \oplus 0^{t-2} \oplus 0$. Since $B_t^-(u)$ and $B_t^-(v)$ contain all vertices that end with 0^{n-t} or $0^{n-t} \oplus 1 \oplus 0^k$ where $k = 0, 1, \ldots, n-t-1$, u and v are t-twins. Thus $\vec{\mathcal{B}}(d,n)$ has no t-identifying code. □

As an additional treat for the reader, we provide a simple construction for 1-identifying codes in $\vec{\mathcal{B}}(d,n)$ whenever we have either $d > 2$ or n odd.

Theorem 4.60. *If n is odd, or n is even and $d > 2$, then*

$$S = \mathcal{A}_d^n \setminus \{a \oplus \mathcal{A}_d^{n-2} \oplus a \mid a \in \mathcal{A}_d\}$$

is an identifying code for $\vec{\mathcal{B}}(d,n)$. Further this identifying code has optimal size $(d-1)d^{n-1}$.

Proof. Define S as in the statement of the theorem. First, we will see that the identifying set for every vertex has size either d or $d-1$. Let $x = x_1x_2\ldots x_n$, then

$$N^-(x) \cap S = \{\mathcal{A} \oplus x_1x_2\ldots x_{n-1}\} \setminus \{x_{n-1}x_1x_2\ldots x_{n-1}\}.$$

If $x_1 = x_n$, then $\mathsf{ID}_S(x) = N^-(x) \cap S$ has size $d-1$. Whereas, if $x_1 \neq x_n$, then $\mathsf{ID}_S(x) = \{x\} \cup N^-(x) \cap S$ has size d.

From this it is clear that every vertex has a non-empty identifying set. However we must also show that every identifying set is unique. Suppose there are two distinct vertices $x, y \in V(\vec{\mathcal{B}}(d,n))$ such that $\mathsf{ID}_S(x) = \mathsf{ID}_S(y)$. Call their identical identifying set T. We look at the two cases, $|T| = d$ and $|T| = d-1$, separately below.

Suppose that $|T| = d$. Then $\{x, y\} \subseteq T$ by our assumption on T and our earlier reasoning. Since $x \neq y$, this means that $\vec{\mathcal{B}}(d, n)$ contains both directed arcs $x \to y$ and $y \to x$. This allows us to conclude that $\{x, y\} = \{(ab)^k, (ba)^k\}$ for some distinct $a, b \in \mathcal{A}$ with $k = n/2$. In particular we must have n even. Below are the precise identifying sets for x and y.

$$\begin{aligned} \mathsf{ID}_S((ab)^k) &= \{(ab)^k, (ba)^k\} \cup \{c(ab)^{k-1}a \mid c \in \mathcal{A} \setminus \{a, b\}\} \\ \mathsf{ID}_S((ba)^k) &= \{(ab)^k, (ba)^k\} \cup \{c(ba)^{k-1}b \mid c \in \mathcal{A} \setminus \{a, b\}\} \end{aligned}$$

If $d > 2$ these two identifying sets are in fact different, which is a contradiction.

Suppose that $|T| = d-1$. Then neither x nor y is in T, which means neither is in S. However since their identifying sets are identical, this means that they have identical first neighborhoods. By definition of first neighborhoods, this means that x and y have the same prefix but different final letters. By then definition of S, one of x, y (if not both) is a member of S, which is a contradiction. □

4.3.2 Undirected de Bruijn Graphs

General Results

We now consider the general undirected de Bruijn graph. Our first result proves the existence of t-identifying codes in $\mathcal{B}(d, n)$ for $d \geq 3$ and relatively large n (with respect to t).

Theorem 4.61. *$\mathcal{B}(d, n)$ is t-identifiable for $d \geq 3$ and $n \geq 2t$.*

To prove this theorem, we will first prove the following lemma, which relies heavily on Lemma 3.34.

Lemma 4.62. *For $n \geq 2t$, the number of distinct t-prefixes in $B_t(y) \setminus [d]^t \oplus y_1 y_2 \dots y_{n-t}$ is at most*

$$1 - d^{\lfloor t/2 \rfloor} + 2 \cdot \sum_{j=0}^{t-1} d^j.$$

Proof. Following Lemma 3.34, the t-prefixes in $B_t(y)$ take one of the following three forms (matching the types in Lemma 3.34).

1. $y_1 y_2 \dots y_t$;
2. $[d]^g \oplus y_{b-f+1} \dots y_{t+b-f-g}$;
3. $[d]^{f-c} \oplus y_{b+1} \dots y_{t+b+c-f}$.

In order to more easily count these t-prefixes, we will sort them by the last letter that appears, and then sort them from longest $[d]^i$ prefix to smallest. Since the largest $[d]^i$ prefix also counts the strings with smaller $[d]^j$ prefix so

long as the strings end in the same letter, this will allow us to count unique prefixes. We begin by rewriting the types of prefixes so as to more easily do this.

1. $y_1 y_2 \dots y_t$;
2. We find this range of y-subsequences by noticing the following:

$$\begin{aligned} \min(b-f) &= (g+1)-(t-2g-1) \\ &= 3g+2-t, \text{ and} \\ \max(b-f) &= \max(g+1, t-g) \\ &= t-g. \end{aligned}$$

   Hence for $0 \le g \le \frac{t-1}{2}$:

$$\begin{aligned} [d]^g &\oplus y_{3g+2-t+1} \cdots y_{2g+2} \\ &\vdots \\ [d]^g &\oplus y_{t-g+1} \cdots y_{2t-2g} \end{aligned}$$

   Last letters: y_i such that $2g+2 \le i \le 2t-2g$.

   Range: y_i is a last letter whenever $t+1 \le i \le 2t$.

   Max g for each i: $\lfloor \frac{2t-i}{2} \rfloor$.

3. Note that in this case, we can cover all cases with $c > 0$ by a different case with $c = 0$, so we may just consider the cases $c = 0$ to simplify things.
   (a) For $0 \le f \le \frac{t+1}{2}$:

$$\begin{aligned} [d]^f &\oplus y_1 \cdots y_{t-f} \\ &\vdots \\ [d]^f &\oplus y_f \cdots y_{t-1} \end{aligned}$$

   Last letters: $y_{\frac{t-1}{2}}, \dots, y_{t-1}$.

   Range: y_i is a last letter whenever $t-f \le i \le t-1$.

   Max f for each i: $\frac{t+1}{2}$.

   (b) For $\frac{t+1}{2} < f < t$ (recall we eliminated $f = t$):

$$\begin{aligned} [d]^f &\oplus y_1 \cdots y_{t-f} \\ &\vdots \\ [d]^f &\oplus y_{t-f+1} \cdots y_{2t-2f} \end{aligned}$$

   Last letters: $y_1, y_2, \dots, y_{t-2}$.

   Range: y_i is a last letter whenever $t-f \le i \le 2t-2f$.

   Max f for each i: $\frac{2t-i}{2}$.

Note that because we require $n \geq 2t$, both cases (2) and (3) cover all possible t-prefixes. That is, we cannot possibly have any t-prefixes that end in $[d]^k$ for any $k > 0$. Additionally, note that each case covers a different range of last letters: (1) $i = t$; (2) $t + 1 \leq i \leq 2t$; and (3) $t - f \leq i \leq t - 1$. Hence we may count each case separately.

1. There is only one string in this case.

2. We showed previously that $\max(g) = \lfloor \frac{2t-i}{2} \rfloor$. Thus we have the following formula.
$$\begin{cases} d^{\frac{t-1}{2}} + 2 \cdot \sum_{j=0}^{\frac{t-3}{2}} d^j, & \text{if } t \text{ is odd;} \\ 2 \cdot \sum_{j=0}^{\frac{t-2}{2}} d^j, & \text{if } t \text{ is even.} \end{cases}$$

3. In this case, our subcases (a) and (b) overlap. We break up our ranges slightly differently this time to determine $\max(f)$.

   (a) $1 \leq i < \frac{t-1}{2}$.
   In this range for i, we must be in the higher range for f, so we have $\max(f) = \lfloor \frac{2t-i}{2} \rfloor$.

   (b) $\frac{t-1}{2} \leq i \leq t - 2$.
   Considering both ranges for f, we have the following maximum value for f, depending on i.
$$\max(f) = \max\left(\frac{t+1}{2}, \left\lfloor \frac{2t-i}{2} \right\rfloor\right) = \left\lfloor \frac{2t-i}{2} \right\rfloor$$

   (c) $i = t - 1$.
   For this value of i, we must be in the lower range for f, and hence we have $\max(f) = \lfloor \frac{t+1}{2} \rfloor = \lfloor \frac{2t-i}{2} \rfloor$.

   Hence all cases (a)-(c) have $\max(f) = \lfloor \frac{2t-i}{2} \rfloor$. Thus we have the following formula.
$$\begin{cases} 2 \cdot \sum_{j=\frac{t+1}{2}}^{t-1} d^j, & \text{if } t \text{ is odd;} \\ -d^{\frac{t}{2}} + 2 \cdot \sum_{j=\frac{t}{2}+1}^{t-1} d^j, & \text{if } t \text{ is even.} \end{cases}$$

Now when we combine all of our equations we get the following final count.
$$1 - d^{\lfloor \frac{t}{2} \rfloor} + 2 \cdot \sum_{j=0}^{t-1} d^j$$

Note that this provides only an upper bound on our t-prefixes - if we have repeated letters than we may have double-counted. □

Now we are ready to prove our theorem.

Proof of Theorem 4.61. Consider two arbitrary strings: $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$. We will show that these two strings cannot be t-twins by showing that $B_t(x) \setminus B_t(y) \neq \emptyset$. This will be done in two cases: $x_1x_2 \dots x_{n-t} \neq y_1y_2 \dots y_{n-t}$ and $x_{t+1}x_{t+2} \dots x_n \neq y_{t+1}y_{t+2} \dots y_n$. Note that this covers all cases, since $x \neq y$ implies there is some $i \in [1, n]$ such that $x_i \neq y_i$. Additionally, since $n \geq 2t$, we must have that $i \in [1, n-t] \cup [t+1, n]$. Hence at least one of these two cases must be true.

1. $x_1x_2 \dots x_{n-t} \neq y_1y_2 \dots y_{n-t}$.

   We will show that there must exist some string in $B_t(x)$ that is not in $B_t(y)$. In particular, there is a string $a \in [d]^t \oplus x_1 \dots x_{n-t}$ such that $a \notin B_t(y)$. We do this by counting the number of distinct t-prefixes in $B_t(y) \setminus [d]^t \oplus y_1y_2 \dots y_{n-t}$, and showing that this number is smaller than d^t. Note that because of the case that we are in, we need not consider the strings in $[d]^t \oplus y_1y_2 \dots y_{n-t}$. If we can show that the number of t-prefixes is smaller than d^t, then there must be some string $z \in B_t(x) \setminus B_t(y)$.

   From Lemma 4.62, we know that the total number of t-prefixes in $\mathcal{B}_t(y) \setminus [d]^t \oplus y_1y_2 \dots y_{n-t}$ is equal to $1 - d^{\lfloor \frac{t}{2} \rfloor} + 2 \cdot \sum_{j=0}^{t-1} d^j$, and that one of those t-prefixes is $y_1 \dots y_t$, which we may ignore because of the case that we are in. Define $f(t) = -d^{\lfloor \frac{t}{2} \rfloor} + 2 \cdot \sum_{j=0}^{t-1} d^j$ and $g(t) = d^t - f(t)$. If we can show that $g(t)$ is always positive for $d \geq 3$, then we know that there exists a string $a \in ([d]^t \oplus x_1 \dots x_{n-t}) \setminus ([d]^t \oplus y_1 \dots y_{n-t}) \subseteq B_t(x) \setminus B_t(y)$. Then we know that x and y are not t-twins.

   Consider our new function $g(t)$.

$$\begin{aligned} g(t) &= d^t + d^{t/2} - 2 \cdot \sum_{j=0}^{t-1} d^j \\ &= d^t + d^{t/2} - \frac{2 \cdot (d^t - 1)}{d - 1} \\ &= \frac{d^t(d-1) + d^{t/2}(d-1) - 2(d^t - 1)}{d - 1} \end{aligned}$$

   We will determine the nature of this function by finding the roots. We find the roots by setting the numerator equal to 0 and making a substitution $x = d^{t/2}$.

$$d^t(d-1) + d^{t/2}(d-1) - 2(d^t - 1) = x^2(d-3) + x(d-1) + 2$$

   The roots of this equation are $x = -1$ and $x = \frac{-4}{2d-6}$. Reversing our substitution this equates to $d^{t/2} = -1$ and $d^{t/2} = \frac{-4}{2d-6}$. The first root is impossible, and the second will only be possible when $2d - 6 < 0$, or $d < 3$. Hence, if $d \geq 3$, our function has no real roots and is always positive.

2. $x_{t+1}x_{t+2} \dots x_n \neq y_{t+1}y_{t+2} \dots y_n$.

In this case, we want to show that there exists some string:

$$a \in \left(x_{t+1} \dots x_n \oplus [d]^t\right) \setminus \left(y_{t+1} \dots y_n \oplus [d]^t\right) \subseteq B_t(x) \setminus B_t(y).$$

Because of the symmetric nature of the strings and edges in the de Bruijn graph, this case follows the same as the previous case, with analogous lemmas to Lemmas 3.34 and 4.62 for t-suffixes (instead of t-prefixes). Thus we will again always have fewer than d^t prefixes represented in $B_t(y) \setminus (y_{t+1} \dots y_n \oplus [d]^t)$, so we will always be able to find the desired string a that can identify x from y.

□

Specific Results

Theorem 4.63. *For $n \geq 3$, the graph $\mathcal{B}(2, n)$ is identifiable.*

Proof. For $n = 3$, the following is a minimum 1-identifying code on $\mathcal{B}(2, 3)$.

$$\{001, 010, 011, 101\}$$

When $n \geq 4$, we have the following proof, with many cases. We will prove this result by showing that it is not possible to have two vertices x and y that are twins. Suppose (for a contradiction) that x and y are in fact twins in $\mathcal{B}(2, n)$. First, the 1-balls for each vertex are as follows.

$$B_1(x) = \left\{ \begin{array}{l} x_1x_2 \dots x_n \\ 0x_1 \dots x_{n-1} \\ 1x_1 \dots x_{n-1} \\ x_2 \dots x_n0 \\ x_2 \dots x_n1 \end{array} \right\} \qquad B_1(y) = \left\{ \begin{array}{l} y_1y_2 \dots y_n \\ 0y_1 \dots y_{n-1} \\ 1y_1 \dots y_{n-1} \\ y_2 \dots y_n0 \\ y_2 \dots y_n1 \end{array} \right\}$$

Without loss of generality, we assume that $x_1 = 0$. Then we have two cases: either $x_1x_2 \dots x_n = 0y_1 \dots y_{n-1}$, or $x_1x_2 \dots x_n \in \{y_2 \dots y_n0, y_2 \dots y_n1\}$.

1. $x_1x_2 \dots x_n = 0y_1 \dots y_{n-1}$.

   In this case, we know that $0x_2 \dots x_n = 0y_1 \dots y_{n-1}$, and so $x_2 \dots x_n = y_1 \dots y_{n-1}$. From this, we know the following equality holds.

   $$\{x_2 \dots x_n0, x_2 \dots x_n1\} = \{y_1y_2 \dots y_n, y_1y_2 \dots \overline{y_n}\}$$

   This gives us two cases: either $y_1y_2 \dots \overline{y_n} \in \{0y_1 \dots y_{n-1}, 1y_1 \dots y_{n-1}\}$, or $y_1y_2 \dots \overline{y_n} \in \{y_2 \dots y_n0, y_2 \dots y_n1\}$.

   (a) $y_1y_2 \dots \overline{y_n} \in \{0y_1 \dots y_{n-1}, 1y_1 \dots y_{n-1}\}$

   The fact that $y_2 \dots \overline{y_n} = y_1 \dots y_{n-1}$ implies the following.

   $$y_1 = y_2 = \dots = y_{n-1} = \overline{y_n}$$

Because we are in Case 1 and $x_2 \dots x_n = y_1 \dots y_{n-1}$, we also have the following equalities.

$$x_2 = x_3 = \cdots = x_n = y_1$$

Hence our 1-balls must be as shown below for some $a \in \{0, 1\}$.

$$B_1(x) = \left\{ \begin{array}{l} 0a \dots a \\ 00a \dots a \\ 10a \dots a \\ a \dots a0 \\ a \dots a1 \end{array} \right\} \qquad B_1(y) = \left\{ \begin{array}{l} a \dots a\overline{a} \\ 0a \dots a \\ 1a \dots a \\ a \dots a\overline{a}0 \\ a \dots a\overline{a}1 \end{array} \right\}$$

Note that since $n \geq 4$, we have two strings in $B_1(y)$ that have different second-to-last and third-to-last letters, however in $B_1(x)$ there are no such strings. Hence these sets cannot possibly be equal, which is a contradiction.

(b) $y_1 y_2 \dots \overline{y_n} \in \{y_2 \dots y_n 0, y_2 \dots y_n 1\}$
This implies that $y_1 y_2 \dots y_{n-1} = y_2 \dots y_n$, and so we have the following chain of equalities.

$$y_1 = y_2 = \cdots = y_{n-1} = y_n$$

Hence $y = a^n$ and $x = 0a^{n-1}$ for some $a \in \{0, 1\}$. Since $x \neq y$, we must have $a = 1$ and thus our 1-balls, given below, are clearly not equal - a contradiction.

$$B_1(x) = \left\{ \begin{array}{l} 01 \dots 1 \\ 001 \dots 1 \\ 101 \dots 1 \\ 1 \dots 10 \\ 1 \dots 11 \end{array} \right\} \qquad B_1(y) = \left\{ \begin{array}{l} 11 \dots 1 \\ 01 \dots 1 \\ 1 \dots 10 \end{array} \right\}$$

2. $x_1 x_2 \dots x_n \in \{y_2 \dots y_n 0, y_2 \dots y_n 1\}$ and $y_2 = 0$.
From this, we have the following 1-balls.

$$B_1(x) = \left\{ \begin{array}{l} 0x_2 \dots x_n \\ 00x_2 \dots x_{n-1} \\ 10x_2 \dots x_{n-1} \\ x_2 \dots x_n 0 \\ x_2 \dots x_n 1 \end{array} \right\} \qquad B_1(y) = \left\{ \begin{array}{l} y_1 0x_2 \dots x_{n-1} \\ 0y_1 0x_2 \dots x_{n-2} \\ 1y_1 0x_2 \dots x_{n-2} \\ 0x_2 \dots x_{n-1} 0 \\ 0x_2 \dots x_{n-1} 1 \end{array} \right\}$$

Now we have two cases: either (a) $1y_1 0x_2 \dots x_{n-2} = 10x_2 \dots x_{n-1}$, or (b) $1y_1 0x_2 \dots x_{n-2} \in \{x_2 \dots x_n 0, x_2 \dots x_n 1\}$.

(a) $1y_1 0x_2 \dots x_{n-2} = 10x_2 \dots x_{n-1}$.
This statement implies that we have the following chain of equalities.

$$y_3 = \cdots = y_n = x_2 = \cdots = x_{n-1}$$

In particular, we now know that $x = 0a \dots a$ and $y = 00a \dots a$. Hence our 1-balls are given below.

$$B_1(x) = \left\{ \begin{array}{l} 0a \dots a \\ 00a \dots a \\ 10a \dots a \\ a \dots a0 \\ a \dots a1 \end{array} \right\} \qquad B_1(y) = \left\{ \begin{array}{l} 00a \dots a \\ 000a \dots a \\ 100a \dots a \\ 0a \dots a0 \\ 0a \dots a1 \end{array} \right\}$$

Since $000a \dots a \in B_1(y)$, the only way to have $B_1(x) = B_1(y)$ would require $a = 0$, and thus $x = y$, which is a contradiction.

(b) $1y_1 0x_2 \dots x_{n-2} \in \{x_2 \dots x_n 0, x_2 \dots x_n 1\}$ and $x_2 = 1$.
In this instance, we know that $x_2 \dots x_n = 1y_1 0x_2 \dots x_{n-3}$, and hence $x_5 \dots x_n = x_2 \dots x_{n-3}$. This tells us that $x = 01y_1 01y_1 \dots$ and $y = y_1 01y_1 01 \dots$. In particular, our 1-balls are now shown below.

$$B_1(x) = \left\{ \begin{array}{l} 01y_1 01y_1 \dots \\ 001y_1 01y_1 \dots \\ 101y_1 01y_1 \dots \\ 1y_1 01y_1 \dots 0 \\ 1y_1 01y_1 \dots 1 \end{array} \right\} \quad B_1(y) = \left\{ \begin{array}{l} y_1 01y_1 01 \dots \\ 0y_1 01y_1 01 \dots \\ 1y_1 01y_1 01 \dots \\ 01y_1 01 \dots 0 \\ 01y_1 01 \dots 1 \end{array} \right\}$$

Note that $B_1(y)$ contains two distinct strings beginning with 01, while $B_1(x)$ contains only one such string. Hence it is not possible that $B_1(x) = B_1(y)$, which contradicts our initial assumption.

□

The binary undirected de Bruijn graph $\mathcal{B}(2, n)$ turns out to be more difficult to establish a set pattern for t-identifiability. Note that we have $\mathcal{B}(2, 10)$ is not 8-identifiable as we would think. We have two pairs of 8-twins:

$$\{0111011110, 0111101110\}$$

and

$$\{1000010001, 1000100001\}.$$

Note that in the *majority* of cases, we find the maximum t such that $\mathcal{B}(2, n)$ is t-identifiable is $t = n - 2$, however there are a few cases in which this does not hold.

For $d \geq 3$, we have a few remaining specific results. Some of these results are also covered by the more general results in the previous section.

Lemma 4.64. *For $n > 3$ and $d \geq 3$, $\mathcal{B}(d, n)$ is 2-identifiable.*

Proof. We will show that for arbitrary $x_1 \dots x_n$ and $y_1 \dots y_n$ we have $B_2(x) \neq B_2(y)$. Recall that we have the following contents in $B_2(y)$.

$$B_2(y_1 \dots y_n) = \left\{ \begin{array}{c} y_1 \dots y_n \\ [d] \oplus y_1 \dots y_{n-1} \\ y_2 \dots y_n \oplus [d] \\ [d]^2 \oplus y_1 \dots y_{n-2} \\ y_3 \dots y_n \oplus [d]^2 \\ [d] \oplus y_2 \dots y_n \\ y_1 \dots y_{n-1} \oplus [d] \end{array} \right\}$$

We have two cases.

1. $x_1 \dots x_{n-2} \neq y_1 \dots y_{n-2}$.

   We will show that there exists some $a_1 \dots a_n \in [d]^2 \oplus x_1 \dots x_{n-2} \subseteq B_2(x)$ that is not in $B_2(y)$. The following table shows the options of how $a_1 \dots a_n$ could lie inside of $B_2(y)$, and the corresponding choices for $a_1 a_2$.

$a_1 \dots a_n = y_1 \dots y_n$	$y_1 y_2$
$a_1 \dots a_n \in [d] \oplus y_1 \dots y_{n-1}$	$[d] \oplus y_1$
$a_1 \dots a_n \in y_2 \dots y_n \oplus [d]$	$y_2 y_3$
$a_1 \dots a_n \in [d]^2 \oplus y_1 \dots y_{n-2}$	N/A
$a_1 \dots a_n \in y_3 \dots y_n \oplus [d]^2$	$y_3 y_4$
$a_1 \dots a_n \in [d] \oplus y_2 \dots y_n$	$[d] \oplus y_2$
$a_1 \dots a_n \in y_1 \dots y_{n-1} \oplus [d]$	$y_1 y_2$

   Note that the fourth row in this table is marked "N/A" because of the case that we are in. Additionally, note that *at most* this list accounts for $2d+2$ of the d^2 possibilities for $a_1 a_2$. Hence whenever $d \geq 3$ there is always a choice for $a_1 a_2$ that does not lie on this list. By choosing that option, we have found $a_1 \dots a_n \in B_2(x) \setminus B_2(y)$.

2. $x_3 \dots x_n \neq y_3 \dots y_n$.

   We will show that there exists some $a_1 \dots a_n \in x_3 \dots x_n \oplus [d]^2 \subseteq B_2(x)$ that is not in $B_2(y)$. The following table shows the options of how $a_1 \dots a_n$ could lie inside of $B_2(y)$, and the corresponding choices for $a_{n-1} a_n$.

$a_1 \dots a_n = y_1 \dots y_n$	$y_{n-1} y_n$
$a_1 \dots a_n \in [d] \oplus y_1 \dots y_{n-1}$	$y_{n-2} y_{n-1}$
$a_1 \dots a_n \in y_2 \dots y_n \oplus [d]$	$y_n \oplus [d]$
$a_1 \dots a_n \in [d]^2 \oplus y_1 \dots y_{n-2}$	$y_{n-3} y_{n-2}$
$a_1 \dots a_n \in y_3 \dots y_n \oplus [d]^2$	N/A
$a_1 \dots a_n \in [d] \oplus y_2 \dots y_n$	$y_{n-1} y_n$
$a_1 \dots a_n \in y_1 \dots y_{n-1} \oplus [d]$	$y_{n-1} \oplus [d]$

   Note that the fifth row in this table is marked "N/A" because of the case that we are in. Additionally, note that *at most* this list accounts for $2d+2$

of the d^2 possibilities for $a_{n-1}a_n$. Hence whenever $d \geq 3$ there is always a choice for $a_{n-1}a_n$ that does not lie on this list. By choosing that option, we have found $a_1 \dots a_n \in B_2(x) \setminus B_2(y)$.

□

Lemma 4.65. *For $d \geq 3$, the graph $\mathcal{B}(d, 3)$ is 2-identifiable.*

Proof. We will show that for arbitrary $x_1x_2x_3$ and $y_1y_2y_3$ we have $B_2(x) \neq B_2(y)$. We have three cases.

1. If $x_1 \neq y_1$, then consider $a_1a_2a_3 \in [d]^2 \oplus x_1 \subseteq B_2(x)$. We will choose a_1a_2 so as to avoid all contents of the following set of size at most $3d-1$: $\{[d] \oplus y_1, [d] \oplus y_2, y_3 \oplus [d], y_1y_2, y_2y_3\}$. Since there are a total of d^2 options for a_1a_2, as long as $d \geq 3$ there is still a choice for a_1a_2 that lies outside of the given set. Once we have selected such a pair a_1a_2, then we have a string $a_1a_2a_3 \in B_2(x) \setminus B_2(y)$.

2. If $x_3 \neq y_3$, then consider $a_1a_2a_3 \in x_3 \oplus [d]^2$. We will choose a_2a_3 so as to avoid all contents of the following set of size at most $3d-1$: $\{[d] \oplus y_1, y_2 \oplus [d], y_3 \oplus [d], y_1y_2, y_2y_3\}$. Since there are a total of d^2 options for a_2a_3, as long as $d \geq 3$ there is still a choice for a_2a_3 that lies outside of the given set. Once we have selected such a pair a_2a_3, then we have a string $a_1a_2a_3 \in B_2(x) \setminus B_2(y)$.

3. Lastly, if $x_1 = y_1$, $x_3 = y_3$, but $x_2 \neq y_2$, then we have several cases.

   (a) If $x_1x_2 = y_2y_3$, then we must have $x = abb$ and $y = aab$ for some $a \neq b \in [d]$. Then $cbb \in B_2(x) \setminus B_2(y)$ for any $c \neq a, b$.

   (b) If $x_1 = x_3$, then $x = aba$ and $y = aca$ for some $b \neq c \in [d]$. At least one of $b, c \neq a$, and so choose some $k \in \{b, c\} \setminus \{a\}$. Then $kak \in B_2(x) \triangle B_2(y)$, and so $B_2(x) \neq B_2(y)$.

   (c) If $x_1x_2 \neq y_2y_3$ and $x_1 \neq x_3$, then we consider $a_1a_2a_3 \in x_1x_2 \oplus [d]$ with $a_3 \neq y_1, y_2$. Then $a_1a_2a_3 \in B_2(x) \setminus B_2(y)$.

Hence in all cases, $B_2(x) \neq B_2(y)$ for arbitrary x, y, so it is not possible to have a pair of 2-twins $\{x, y\}$ in $\mathcal{B}(d, 3)$. Thus $\mathcal{B}(d, 3)$ is 2-identifiable. □

Note that Lemmas 4.64 and 4.65 combined tell us that all graphs $\mathcal{B}(d, n)$ with $d, n \geq 3$ are 2-identifiable.

Bounds

Theorem 4.66 ([14])**.** *The size of an identifying code for a regular graph with N vertices and vertex degree D is lower-bounded by*

$$M(t) \geq \frac{2N}{D+2}.$$

Proof. Consider the $K \times N$ binary matrix A where $a_{kn} = 1$ if and only if the kth codeword covers the nth vertex, and $a_{kn} = 0$ otherwise. There must be $K(D+1)$ nonzero entries in the matrix, as each codeword covers $D+1$ vertices. On the other hand, at most K columns can have weight 1, while the remaining $N-K$ columns must have weight at least 2, the number of nonzero entries must be at least $K + 2(N-K) = 2N - K$. Thus $K(D+1) \geq 2N - K$, or

$$M(t) = K \geq \frac{2N}{D+2}.$$

□

Corollary 4.67. *The size of an identifying code for a graph with N vertices and maximum degree D is lower-bounded by*

$$M(t) \geq \frac{2N}{D+2}.$$

Proof. Using the previous argument, the maximum number of nonzero entries in the matrix is $K(D+1)$. The rest of the argument remains unchanged. □

Note that for the undirected de Bruijn graph, the multiple edges and loops do not change the identifying codes from the underlying simple, undirected graph. Thus we can apply Corollary 4.67 easily by simply removing multiple edges and loops and considering a graph with maximum degree $2d$.

Corollary 4.68. *The size of an identifying code for the undirected binary de Bruijn graph $\mathcal{B}(2,n)$ is lower-bounded by*

$$M(t) \geq \frac{2^{n+1}}{6} = \frac{1}{3} \cdot |\mathcal{B}(2,n)|.$$

Corollary 4.69. *The size of an identifying code for the undirected de Bruijn graph $\mathcal{B}(d,n)$ is lower-bounded by*

$$M(t) \geq \frac{d^n}{d+1}.$$

In our small examples of $\mathcal{B}(2,3)$ and $\mathcal{B}(2,4)$, this bound agrees. For example, a minimum 1-identifying code for $\mathcal{B}(2,3)$ is $\{001, 010, 011, 101\}$ with size 4. Corollary 4.68 gives us a lower bound of $\frac{8}{3} = 2.6\overline{6}$. Also, a minimum 1-identifying code for $\mathcal{B}(2,4)$ is $\{0001, 0010, 0101, 0111, 1011, 1100\}$ with size 6, while Corollary 4.68 gives us a lower bound of $\frac{16}{3} = 5.3\overline{3}$, a tight bound in this case.

An alternative lower bound is given by the following theorem that is based on the fact that the de Bruijn graph is a line graph. For more on this, see Chapter 4.5.

Theorem 4.70 ([11])**.** *Let G be a twin-free line graph on $n \geq 4$ vertices. Then we have*

$$\gamma^{\mathsf{ID}}(G) \geq \frac{3\sqrt{2}}{4}\sqrt{n}.$$

For the de Bruijn graph, this implies a lower bound of $\frac{3\sqrt{2}}{4}\sqrt{d^n}$. However, the bound from Corollary 4.69 gives us a better bound for this class of graphs.

Non-Optimal Constructions

The following result is inspired by an identical result for the n-dimensional binary cube in [14].

Theorem 4.71. *If C^* is an optimal 2-identifying code for the undirected de Bruijn graph $\mathcal{B}(d,n)$, then*

$$C = \{w \mid \exists v \in C^* \text{ s.t. } d(v,w) = 1\}$$

is a 1-identifying code.

Proof. We will show that every vertex in the undirected de Bruijn graph $\mathcal{B}(d,n)$ is covered by a unique set of codewords. Let $x = x_1x_2\dots x_n \in \mathcal{B}(d,n)$. Note that the vertices at distance 1 from x are:

$$\begin{array}{rclcrcl} U_0 & = & x_2x_3\dots x_n0 & & V_0 & = & 0x_1x_2\dots x_{n-1} \\ U_1 & = & x_2x_3\dots x_n1 & & V_1 & = & 1x_1x_2\dots x_{n-1} \\ U_2 & = & x_2x_3\dots x_n2 & & V_2 & = & 2x_1x_2\dots x_{n-1} \\ \vdots & & & & \vdots & & \\ U_{d-1} & = & x_2x_3\dots x_n(d-1) & & V_{d-1} & = & (d-1)x_1x_2\dots x_{n-1} \end{array}$$

We will refer to vertices of type U_i as undirected out-neighbors and vertices of type V_i as undirected in-neighbors for obvious reasons. We have three cases.

Case 1: $x \in C^*$: In this case, x is covered by all U_i, V_i for $i \in \{0,1,2,\dots,d-1\}$ for the code C. If some $w = w_1w_2\dots w_n$ is also covered by all d U_i's and all d V_i's, then we must prove that $w = x$.

Note that for such a vertex w, we can only have that w is an undirected out-neighbor of at most one U_i and is an undirected in-neighbor of at most one V_j. Thus we have that w is the in-neighbor of at least one U_i and the out-neighbor of at least one V_j. Then we know some information about the letters of w, namely:

$$\begin{array}{rcl} w & = & x_1x_2\dots x_{n-1}w_n \\ & = & w_1x_2\dots x_{n-1}x_n. \end{array}$$

Hence we must have $w_1 = x_1$ and $w_n = x_n$, or in other words $w = x$.

Case 2: $x \in C \setminus C^*$: In this case, x is covered by itself and every one of its neighbors in C. The only other vertices covered by x are the vertices U_i, V_i for $i \in \{0,1,2,\dots,d-1\}$. We must show that x is covered by something that each of its neighbors is not covered by (or vice versa), thus proving that they have different cover sets.

Let $T \in \{U_i \mid i \in \{0, 1, \ldots, d-1\} \cup \{V_i \mid i \in \{0, 1, \ldots, d-1\}$ be arbitrary. Since $x \in C$, there must be some $y \in C^*$ with $d(y, x) = 1$. This implies that $d(y, T) \leq 2$. Then in the code C^*, both x and T are covered by y, since C^* has radius $t = 2$. However since C^* is also an identifying code, x and T must have different identifying sets, so there is some $z \in C^*$ that either:

1. covers x and not T, or
2. covers T and not x.

In Case (2.1), we must have $d(z, x) \leq 2$ and $d(z, T) > 2$, which implies that $d(z, x) = 2$. Then there must be some a between z and x, i.e. $d(z, a) = 1$ and $d(x, a) = 1$. Then $a \in C$ and also covers x for code C, but a can't cover T for C as otherwise we would have T covered by z for C^*. See Figure 4.

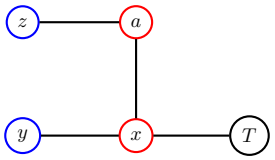


Figure 4: Case 2.1: Blue nodes in $C*$, red nodes in C.

In Case (2.2), we must have $d(z, T) \leq 2$ and $d(z, x) > 2$, which implies that $d(z, T) = 2$. Then there is some a between z and T, i.e. $d(z, a) = 1$ and $d(a, T) = 1$. This implies that $a \in C$ and covers T for C, but a can't cover x for C as otherwise we would have x covered by z for C^*. See Figure 5.

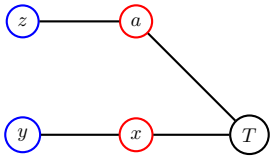


Figure 5: Case 2.2: Blue nodes in $C*$, red nodes in C.

Case 3: $x \in V \setminus (C \cup C^*)$**:** In this case, x is covered by some $y \in C$ and some $z \in C^*$ with $d(z, y) = 1$. We will compare x with another vertex $w \in V$ that is covered by y for C and show that they have different identifying sets for C. We first note that we must have $d(w, y) \leq d(x, y) = 1$ and $d(w, z) \leq d(x, z) = 2$, so w is also covered by z for C^*. Since C^* is an identifying code, there must be some $v \in C^*$ that either:

1. covers x and not w, or
2. covers w and not x.

See Figure 6.

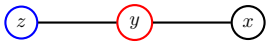


Figure 6: Case 3: Blue nodes in $C*$, red nodes in C.

In Case (1), since $x \notin C$, we have $d(v, x) = 2$ and there is some $a \in C$ between v and x. As v does not cover w for C^*, we must have $d(a, w) > 1$, so a covers x but not w for C. See Figure 7.

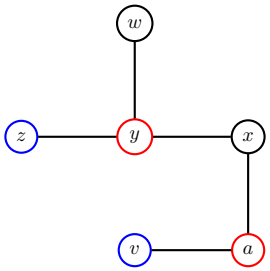


Figure 7: Case 3.1: Blue nodes in $C*$, red nodes in C.

For Case (2), since $x \notin C$, we have $d(v, x) > 1$. Also, as v covers w but not x for C^*, we must have $1 \leq d(w, v) \leq 2$.

If $d(w, v) = 1$, then $w \neq y$, as otherwise v covers x for C^*. This implies that $d(w, x) > 1$, else v would cover x for C^*. So we must have $w \in C$, hence w covers itself but not x for C. See Figure 8.

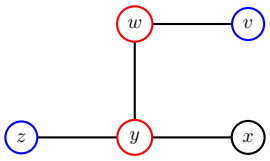


Figure 8: Case 3.2.1: Blue nodes in $C*$, red nodes in C.

Finally, if $d(w, v) = 2$, then there must be some $a \in C$ with $d(a, w) = 1 = d(a, v)$. This implies that $d(a, x) > 1$, since otherwise we would have v covering x for C^*. Hence a covers w but not x for C. See Figure 9.

□

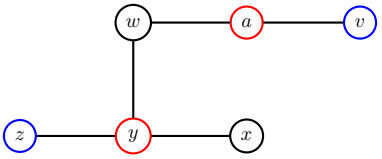


Figure 9: Case 3.2.2: Blue nodes in $C*$, red nodes in C.

Next we would like to consider constructions using direct sums. This idea comes from constructions developed for the cube graph Q_n with edges between strings with Hamming distance one (strings differ in any one bit). For example, results similar to the following two theorems would be nice.

Theorem 4.72 ([5])**.** *Assume that C is 1-identifying on Q_n. Then the direct sum $\{0,1\} \oplus C$ is 1-identifying on Q_{n+1} if and only if $d(\mathbf{c}, C \setminus \{\mathbf{c}\}) \leq 1$ for all $\mathbf{c} \in C$.*

Theorem 4.73 ([5])**.** *If C is 1-identifying on Q_n then $C \oplus \{00, 01, 10, 11\}$ is 1-identifying on Q_{n+2}.*

A first thought might be to consider $C^* = \{0,1\} \oplus C \oplus \{0,1\}$, where C is a 1-identifying code on $\mathcal{B}(2, n-2)$. However, if the vertex $0^{n-2} \in V(\mathcal{B}(2, n-2))$ is in C and is only covered by itself, then we have:

$$\begin{aligned} B_1(10^{n-1}) \cap C^* &= \{0^n, 10^{n-1}, 0^{n-1}1, 10^{n-2}1\} \\ B_1(0^{n-1}1) \cap C^* &= \{0^n, 10^{n-1}, 0^{n-1}1, 10^{n-2}1\} \end{aligned}$$

Note that $B_1(10^{n-1}) \cap C^*$ is missing only the neighbor 110^{n-2}, as if this vertex were included then we must have $10^{n-3} \in C$, which would cover 0^{n-2}. Likewise, $B_1(0^{n-1}1) \cap C^*$ is missing $0^{n-2}11$ because we cannot have $0^{n-3}1 \in C$.

An initial result similar to these is the following.

Theorem 4.74. *Let C' be a 1-identifying code on $\mathcal{B}(2, n-1)$ such that for every $x \in V(\mathcal{B}(2, n-1))$ we have at least one undirected in-neighbor of x in C', and at least one undirected out-neighbor of x in C'. Then $C = \{0,1\} \oplus C'$ is a 1-identifying code on $\mathcal{B}(2, n)$.*

Proof. We must show that C satisfies the following two conditions: (1) every $x \in V(\mathcal{B}(2, n))$ has a neighbor in C, and (2) for every $x, y \in V(\mathcal{B}(2, n))$ we have $\mathsf{ID}_C(x) \neq \mathsf{ID}_C(y)$.

1. Let $x = x_1 x_2 \dots x_n \in V(\mathcal{B}(2, n))$. Then we must have that $x' = x_2 x_3 \dots x_n$ is covered by some undirected out-neighbor $z' = x_3 x_4 \dots x_n z_n \in C'$. Then we must have

   $$\{0 x_3 x_4 \dots x_n z_n, 1 x_3 x_4 \dots x_n z_n\} = \{0,1\} \oplus z' \subseteq C.$$

   In other words, we have $x_2 x_3 \dots x_n z_n \in C$, which is also a neighbor of x, so x is covered.

2. Let $x = x_1x_2 \dots x_n, y = y_1y_2 \dots y_n \in V(\mathcal{B}(2,n))$ be distinct. Define $x' = x_2x_3 \dots x_n$ and $y' = y_2y_3 \dots y_n$. We have two cases.

   $\mathbf{x' = y'}$: In this case we must have $y_1 = \overline{x_1}$. Let $w' \in C'$ be an undirected in-neighbor to $x' = y'$. Then we must have $w_3 \dots w_n = x_2 \dots x_{n-1}$, and hence

   $$w_2w_3 \dots w_n \in \{x_1x_2 \dots x_{n-1}, \overline{x_1}x_2 \dots x_{n-1}\}.$$

   Thus we must have that $0 \oplus w'$ and $1 \oplus w'$ are elements in C and also undirected in-neighbors of *either* x or y, but not both. Therefore $w \in \mathsf{ID}_C(x) \triangle \mathsf{ID}_C(y)$, so $\mathsf{ID}_C(x) \neq \mathsf{ID}_C(y)$.

   $\mathbf{x' \neq y'}$: Without loss of generality we may assume that there is some $w' \in \mathsf{ID}_{C'}(x') \setminus \mathsf{ID}_{C'}(y')$. This implies that both $w' \not\to y'$ and $y' \not\to w'$. In other words, we know that both $w_3 \dots w_n \neq y_2 \dots y_{n-1}$ and $w_2 \dots w_{n-1} \neq y_3 \dots y_n$. Hence $y_1y_2 \dots y_{n-1} \notin \{0,1\} \oplus w_3 \dots w_n$ and $y_2y_3 \dots y_n \notin \{0,1\} \oplus w_2 \dots w_{n-1}$, and therefore $w \not\to y$ and $y \not\to w$. Thus $w \in \mathsf{ID}_C(x) \setminus \mathsf{ID}_C(y)$, so the identifying sets are distinct.

□

While the requirement that all codewords have both in- and out-neighbors in the identifying code is a much stronger requirement than those in Theorem 4.72, it does still provide identifying codes of small cardinality. For example, the minimum size identifying code in the graph $\mathcal{B}(2,3)$ has size 4, however no identifying codes of size 4 are extendable under this operation. Of the 18 identifying codes of size 5, 8 are extendable to $\mathcal{B}(2,4)$, and two of the 8 satisfy the conditions of Theorem 4.74. These two graphs are shown in Figure 10.

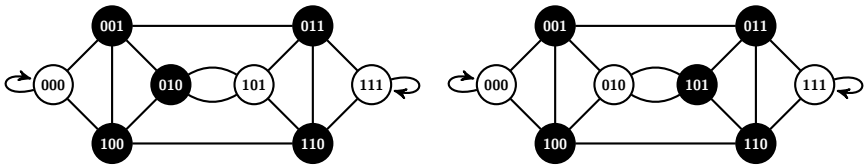


Figure 10: Identifying Codes for $\mathcal{B}(2,3)$ of size 5 satisfying our conditions 4.74

For certain cases, building identifying codes is much simpler. We have the following two theorems for the graphs $\mathcal{B}(d,2)$ for all d. Note that the first theorem allows for exactly one empty identifying set.

Theorem 4.75. *The set $S = \{0i, i0 \mid i \in [d-2]\}$ is an identifying code for $\mathcal{B}(d,2)$.*

Proof. First, for $ab \in V(\mathcal{B}(d,2))$, we have the following identifying sets $\mathsf{ID}(ab)$.

1. If $ab = 00$, then $\mathsf{ID}(ab) = S$.

2. If $ab = 0b$ for $b \in [d-2]$, then $\mathsf{ID}(ab) = \{i0 \mid i \in [d-2]\} \cup \{0b\}$.

3. If $ab = 0(d-1)$, then $\mathsf{ID}(ab) = \{i0 \mid i \in [d-2]\}$.

4. If $ab = a0$ for $a \in [d-2]$, then $\mathsf{ID}(ab) = \{0i \mid i \in [d-2]\} \cup \{a0\}$.

5. If $ab = (d-1)0$, then $\mathsf{ID}(ab) = \{0i \mid i \in [d-2]\}$.

6. If $ab = ab$ for $a, b \in [d-2]$, then $\mathsf{ID}(ab) = \{0a\} \cup \{b0\}$.

7. If $ab = a(d-1)$ for $a \in [d-2]$, then $\mathsf{ID}(ab) = \{0a\}$.

8. If $ab = (d-1)b$ for $b \in [d-2]$, then $\mathsf{ID}(ab) = \{b0\}$.

9. If $ab = (d-1)(d-1)$, then $\mathsf{ID}(ab) = \{\}$.

Next, we must show that for any distinct pair $ab, xy \in V(\mathcal{B}(d,2))$, $\mathsf{ID}(ab) \neq \mathsf{ID}(xy)$. Clearly if ab and xy are of different types, then $\mathsf{ID}(ab) \neq \mathsf{ID}(xy)$. Also, types (1), (3), (5), and (9) have only one element. That only leaves us with the following five cases.

2. We must have $0b \neq 0y$. Then since $0b \in \mathsf{ID}(0b) \setminus \mathsf{ID}(0y)$, we know that $\mathsf{ID}(0b) \neq \mathsf{ID}(0y)$.

4. We must have $a0 \neq x0$. Then since $a0 \in \mathsf{ID}(a0) \setminus \mathsf{ID}(x0)$, we know that $\mathsf{ID}(a0) \neq \mathsf{ID}(x0)$.

6. We must have either $a \neq x$, and so $0a \in \mathsf{ID}(ab) \setminus \mathsf{ID}(xy)$, or $b \neq y$ and hence $b0 \in \mathsf{ID}(ab) \setminus \mathsf{ID}(xy)$. In either case, we have $\mathsf{ID}(ab) \neq \mathsf{ID}(xy)$.

7. We have $a(d-1) \neq x(d-1)$. Then since $0a \in \mathsf{ID}(a(d-1)) \setminus \mathsf{ID}(x(d-1))$, we know that $\mathsf{ID}(a(d-1)) \neq \mathsf{ID}(x(d-1))$.

8. We have $(d-1)b \neq (d-1)y$. Then since $b0 \in \mathsf{ID}((d-1)b)) \setminus \mathsf{ID}((d-1)y)$, we know that $\mathsf{ID}((d-1)b) \neq \mathsf{ID}((d-1)y)$.

□

The previous theorem gives us an identifying code for $\mathcal{B}(d,2)$ of size $2(d-2)$. The next theorem illustrates an identifying code for $\mathcal{B}(d,2)$ of size $\lfloor \frac{3d}{2} \rfloor$, which is an improvement over the last result whenever $d > 8$.

Theorem 4.76. *Define the following sets.*

$$\begin{aligned} S &= \{12, 23, 34, \ldots, (d-1)d, d1\} \\ T &= \begin{cases} \{13, 35, 57, \ldots, (d-2)d\}, & \text{if } d \text{ is odd;} \\ \{13, 35, 57, \ldots, (d-1)1\}, & \text{if } d \text{ is even.} \end{cases} \end{aligned}$$

Then $S \cup T$ is an identifying code for $\mathcal{B}(d,2)$.

Proof. First, we show that every vertex in $\mathcal{B}(d,2)$ is covered by S. For any $ab \in \mathcal{B}(d-2)$, ab is adjacent to both $b(b+1)$ and $(a-1)a$.

Next, we must show that all identifying sets are unique. Note that if a vertex ab has only one neighbor in S, then $a = b+1$ so $ab = (b+1)b$. Note that for all b, there is exactly one vertex $b(b+1)$ with $|N[(b+1)b] \cap S| = 1$, so the identifying sets are unique.

Otherwise, we assume $ab \neq (b+1)b$ and $xy \neq (y+1)y$ with $ab \neq xy$ and show that $N[xy] \cap T \neq N[ab] \cap T$. Note that if $N[ab] \cap S \neq N[xy] \cap S$ then we are done, so we assume otherwise. Then we have $\{b(b+1), (a-1)a\} = \{y(y+1), (x-1)x\}$. This gives us two cases.

1. $b(b+1) = y(y+1)$ and $(a-1)a = (x-1)x$.

   In this case we have $b = y$ and $a = x$, or $ab = xy$, which is a contradiction.

2. $b(b+1) = (x-1)x$ and $(a-1)a = y(y+1)$.

   In this case we have $b = x-1$ and $a = y+1$. In other words, we have $ab = (y+1)(x-1)$. We have two subcases.

   2.1 If x is odd, then $x-1$ is even. Then $T \cap N[xy]$ contains $(x-2)x$. Note that $N[(y+1)(x-1)] \cap T$ contains $(x-2)x$ only if $x = y+1$, which is a contradiction. Thus we must have

   $$(x-2)x \in (N[xy] \cap T) \setminus (N[(y+1)(x-1)] \cap T.$$

   2.2 If x is even, then $x-1$ is odd. Then $N[(y+1)(x-1)] \cap T$ contains $(x-1)(x+1)$. Note that $N[xy] \cap T$ contains $(x-1)(x+1)$ only if $y = x-1$, or $x = y+1$, which is a contradiction. Thus we must have

   $$(x-1)(x+1) \in (N[(y+1)(x-1)] \cap T) \setminus (N[xy] \cap T).$$

□

4.4 De Bruijn Functions

4.4.1 Distance

Many parameters that we are interested in rely on distance in de Bruijn graphs. Fortunately, formulas for distance in both the directed and undirected graphs have already been determined.

Theorem 4.77 ([17])**.** *For all X, Y in the directed graph $\vec{\mathcal{B}}(d,n)$,*

$$d(X,Y) = n - \max\{s \mid 1 \leq s \leq n, x_{n-s+1}x_{n-s+2}\ldots x_n = y_1y_2\ldots y_s$$

where, by convention, the maximum over an empty set is zero.

The formula for the directed graph is straightforward and simply computes the maximum match of suffix in X and prefix in Y.

Theorem 4.78 ([17]). *For all X, Y in the undirected graph $\mathcal{B}(d,n)$,*

$$d(X,Y) = 2n-1+\min\{\min_{1\le i,j\le n}(i-j-\ell_{i,j}(X,Y)), \min_{1\le i,j\le n}(-i+j-r_{i,j}(X,Y))\}$$

where

$$\begin{aligned}\ell_{i,j}(X,Y) &= \max\{s \mid s\le j, s\le n-i+1,\\ &\quad x_i x_{i+1}\dots x_{i+s-1} = y_{j-s+1}y_{j-s+2}\dots y_j\\ r_{i,j}(X,Y) &= \max\{s \mid s\le i, s\le k-j+1,\\ &\quad x_{i-s+1}x_{i-s+2}\dots x_i = y_j y_{j+1}\dots y_{j+s-1}\end{aligned}$$

where, by convention, the maximum over an empty set is zero.

The following example illustrates the mechanics of this formula for the undirected graph. Let $X = 010$ and $Y = 110$ in the undirected de Bruijn graph $\mathcal{B}(2,3)$. Then, using Liu's algorithm, we must compute the following.

$$\begin{aligned}D(X,Y) &= 2n-1+\min_{1\le i,j\le n}(i-j-\max\{\ell_{i,j}(X,Y), r_{j,i}(X,Y)\})\\ &= 5+\min_{1\le i,j\le 3}(i-j-\max\{\ell_{i,j}(X,Y), r_{j,i}(X,Y)\})\end{aligned}$$

Our functions $\ell_{i,j}$ and $r_{j,i}$ are as follows.

$$\begin{aligned}\ell_{i,j} &= \max\{s \mid s\le j, s\le 4-i, x_i x_{i+1}\dots x_{i+s-1} = y_{j-s+1}y_{j-s+2}\dots y_j\}\\ r_{j,i} &= \max\{s \mid s\le j, s\le 4-i, x_{j-s+1}x_{j-s+2}\dots x_j = y_i y_{i+1}\dots y_{i+s-1}\}\end{aligned}$$

We find the following values.

i	j	$\ell_{i,j}(X,Y)$	$r_{j,i}(X,Y)$	$\max\{\ell_{i,j}, r_{j,i}\}$	$i-j-\max\{\ell_{i,j}, r_{j,i}\}$
1	1	$\{\} = 0$	$\{\} = 0$	0	0
1	2	$\{\} = 0$	$\{1\} = 1$	1	-2
1	3	$\{1\} = 1$	$\{\} = 0$	1	-3
2	1	$\{1\} = 1$	$\{\} = 0$	1	0
2	2	$\{1\} = 1$	$\{1\} = 1$	1	-1
2	3	$\{2\} = 2$	$\{2\} = 2$	2	-3
3	1	$\{\} = 0$	$\{1\} = 1$	1	1
3	2	$\{\} = 0$	$\{\} = 0$	0	1
3	3	$\{1\} = 1$	$\{1\} = 1$	1	-1

The minimum in the right-most column is -3, and so we find $D(X,Y) = 5+(-3) = 2$.

We have implemented these functions in the following manner using Matlab.

We propose the following conjecture. Define the term **distance class t for vertex X** as the set

$$D_t(X) = \{Y \mid d(X,Y) = t\}.$$

Conjecture 4.79. *The set $V \setminus \{D_0(0^n), D_1(0^n), D_n(0^n)\}$ is an identifying code for $\mathcal{B}(2,n)$ when $n \ge 4$.*

Algorithm 1 LHS(i, j, X, Y): Computing $\ell_{i,j}(X, Y)$

procedure LHS(i, j, X, Y)
    $n = \text{length}(X)$
    sMax $= 0$
    $s = 0$
    while $(s \leq j)$ **and** $(s \leq n - i + 1)$ **do**
        if $x_i x_{i+1} \dots x_{i+s-1} = y_{j-s+1} y_{j-s+2} \dots y_j$ **then**
            sMax $= s$
        end if
        $s = s + 1$
    end while
end procedure
return sMax

Algorithm 2 RHS(i, j, X, Y): Computing $r_{i,j}(X, Y)$

procedure RHS(i, j, X, Y)
    $n = \text{length}(X)$
    sMax $= 0$
    $s = 0$
    while $(s \leq i)$ **and** $(s \leq n - j + 1)$ **do**
        if $x_{i-s+1} x_{i-s+2} \dots x_i = y_j y_{j+1} \dots y_{j+s-1}$ **then**
            sMax $= s$
        end if
        $s = s + 1$
    end while
end procedure
return sMax

Algorithm 3 D(X, Y): Computing the distance between X and Y

procedure D(X, Y)
    $n = \text{length}(X)$
    $M = \text{zeros}(n, n)$
    for i from 1 to n **do**
        for j from 1 to n **do**
            $M(i, j) = i - j - \max\{\text{LHS}(i, j, X, Y), \text{RHS}(j, i, X, Y)\}$
        end for
    end for
    distance $= 2n - 1 + \min(\min(M), [\,], 2)$
end procedure
return distance

While the sets $D_0(0^n)$ and $D_1(0^n)$ are easily determined:

$$\begin{aligned} D_0(0^n) &= \{0^n\}, \text{ and} \\ D_1(0^n) &= \{10^{n-1}, 0^{n-1}1\}, \end{aligned}$$

the set $D_n(0^n)$ is a more difficult computation. In order to determine this set, we must determine $X \in \mathcal{B}(2,n)$ such that $d(X,0^n) = n$. As a start, we direct the interested reader to Lemma 3.35, which shows that in the nonbinary case for a specific node there is always at least one other node at distance n.

4.4.2 Balls in Directed de Bruijn Graphs

We begin by exploring formulas for $B_t^-(x)$.

Lemma 4.80. $B_t^-(x_1x_2\dots x_n) = \bigcup_{i=0}^{t} \mathcal{A}_d^i \oplus x_1x_2\dots x_{n-i}$.

Definition 4.81. We will refer to the set $\mathcal{A}_d^i \oplus x_1x_2\dots x_{n-i}$ from Lemma 4.80 as suffix class i with respect to x. We will denote this by $\mathcal{S}_i(x)$.

Lemma 4.82. $|\mathcal{S}_i(x)| = d^i$.

Definition 4.83. If there exist $i,j \in [0,t]$ with $i < j$ such that $x_1x_2\dots x_{n-j} = x_{j-i+1}x_{j-i+2}\dots x_{n-i}$, then we will say that x has an (i,j)-shift.

The following lemma makes clear why we chose this notation.

Lemma 4.84. *If there exists a string y and $i,j \in [0,t]$ with $i < j$ such that $y \in \mathcal{S}_i(x)$ and $y \in \mathcal{S}_j(x)$, then x has an (i,j)-shift.*

Proof. We know that the following two equalities must be true.

$$\begin{aligned} y_{i+1}y_{i+2}\dots y_n &= x_1x_2\dots x_{n-i} \\ y_{j+1}y_{j+2}\dots y_n &= x_1x_2\dots x_{n-j} \end{aligned}$$

Since $i < j$, note that the second equality compares shorter strings. Thus we deduce that the following equalities must also hold.

$$\begin{array}{ccccc} x_{n-i} & = & y_n & = & x_{n-j} \\ x_{n-i-1} & = & y_{n-1} & = & x_{n-j-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{j-i+1} & = & y_{j+1} & = & x_1 \end{array}$$

Thus we have $x_1x_2\dots x_{n-j} = x_{j-i+1}x_{j-i+2}\dots x_{n-i}$, as required. □

Lemma 4.85. *If $x = x_1x_2\dots x_n$ has an (i,j)-shift, then $\mathcal{S}_i(x) \subseteq \mathcal{S}_j(x)$.*

Proof. Reversing the argument from Lemma 4.84, we have that an (i,j)-shift implies that

$$y_{i+1}y_{i+2}\dots y_n = x_1x_2\dots x_{n-i}.$$

Since there are no restrictions on $y_1y_2\dots y_i$, we see that *any* string that satisfies these restrictions on $y_{i+1}y_{i+2}\dots y_n$ is in both suffix classes. This describes all strings in suffix class i, and hence we must have $\mathcal{S}_i(x) \subseteq \mathcal{S}_j(x)$. □

Combining the last two lemmas, we arrive at the more complete version.

Lemma 4.86. *String x has an (i,j)-shift if and only if $\mathcal{S}_i(x) \subseteq \mathcal{S}_j(x)$.*

Lemma 4.87. *If x has no (i,j)-shifts, then*

$$\left|B_t^-(x)\right| = \left|\bigcup_{i=0}^{t} \mathcal{S}_i(x)\right| = \sum_{i=0}^{t} d^i = \frac{1-d^{t+1}}{1-d}.$$

Lemma 4.88. *If x admits an (i,j)-shift, then there are d^i double-counted strings in $B_t^-(x)$.*

Proof. By Lemma 4.87, if there are no (i,j)-shifts, then $|B_t^-(x)| = \sum_{i=0}^{t} d^i$. By Lemma 4.86, there is a bijection between (i,j)-shifts and nested suffix classes as follows.

$$(i,j) \mapsto (\mathcal{S}_i(x), \mathcal{S}_j(x))$$

In other words, each (i,j)-shift corresponds to every string in $\mathcal{S}_i(x)$ also being counted in $\mathcal{S}_j(x)$. Thus, for counting purposes, we have d^i double-counted strings. □

Lemma 4.89. *If x admits an (i,j)-shift and a (j,k)-shift, then it admits an (i,k)-shift.*

Proof. By Lemma 4.86, an (i,j)-shift implies $\mathcal{S}_i(x) \subseteq \mathcal{S}_j(x)$, and a (j,k)-shift implies $\mathcal{S}_j(x) \subseteq \mathcal{S}_k(x)$. Hence we must also have $\mathcal{S}_i(x) \subseteq \mathcal{S}_k(x)$, which corresponds to an (i,k)-shift. □

Theorem 4.90.

$$\left|B_t^-(x)\right| = \left(\sum_{i=0}^{t} d_i\right) - \left(\sum_{\substack{i\in[0,t-1] \\ \exists (i,j)\text{-shift for some } j}} d^i\right).$$

Proof. By Lemma 4.87, we start with the total number of strings (including multiplicities) at $\sum_{i=0}^{t} d_i$. By Lemma 4.88, we need to subtract d^i when there exists an (i,j)-shift, but only subtract once for each i. Thus we arrive at the given formula. □

4.4.3 Other Useful Functions and Matlab

In this section we present some de Bruijn functions that return many useful parameters. Although the following functions are useful enough to warrant their inclusion in this report, they do not, taken individually, justify their own dedicated section. Therefore, we opted to present them all in this section. The first of these useful functions that we present is one called *GenerateNodes*. This function returns the string representation in the desired base.

Algorithm 4 GenerateNodes: Generates all strings of length n and base d

```
1: procedure GenerateNodes(d, n)
2:     N = d^n
3:     nodes = cell(1, n)
4:     for i from 1 to N do
5:         nodes{i} = dec2base(i − 1, d, n)
6:     end for
7: end procedure
8: return nodes
```

Notice that in line 5 the MATLAB *string library* function *dec2base*$(i-1, d, n)$ is called. This is where the true work gets done in this function. This method converts the value $i-1$ from a decimal number to one of base n. As an example we executed *GenerateNodes(3, 2)* in MATLAB, and the output is provided below.

$$\text{ans } = \{00, 01, 02, 10, 11, 12, 20, 21, 22\}$$

An alternative method for representing a graph, rather than in the vertex and arc graph that has been used up until now in this report, is to use an N–by–N matrix, where N is the number of vertices in the graph. For example an *adjacency matrix* for a graph is used to represent which vertices in a graph are adjacent to which other vertices. The code to produce an adjacency matrix for $\vec{\mathcal{B}}(d, n)$ follows.

Algorithm 5 AdjacencyMat: Generates Directed Adjacency Matrix

```
1: procedure AdjacencyMat(d, n)
2:     N = d^n
3:     for i from 1 to N do
4:         x = dec2base(i − 1, d, n)
5:         for j from 0 to d − 1 do
6:             y = strcat(x(2 : n),num2str(j))
7:             z = base2dec(y, d)
8:             A(i, z + 1) = 1
9:         end for
10:    end for
11: end procedure
12: return A
```

There are a few more MATLAB *string library* functions used in *AdjacencyMat*. In line 4 we see the function *dec2base(decimal, base, length)* again. In this case, the function converts decimal strings to a base system specified in the third parameter. Function *base2dec()* in line 7 performs the opposite operation. The function *num2string(j)* converts a number j into a string. The function *strcat*$(x, (2 : n))$ is used to concatenate string x removing string positions 2 onward to the tail. Essentially what this function does is revealed in line 8 where

matrix A gets re–populated when adjacent nodes are found. That is, it visits every row i placing a 1 at every column positioned $z + 1$ (N, the number of nodes minus d, the degree) distance apart. When this method is executed using the following command (as an example) in MATLAB "AdjacencyMat(2, 4)", the adjacency matrix in Table 4.1 is returned.

To read the MATLAB output for *AdjacencyMat(2, 4)* (in Table 4.1 above), select a vertex string identifier (name) from the row label (highlighted blue) and then scan across the row associated with the vertex. If a 1 appears in the row then the vertex listed in the corresponding column label (also highlighted blue) is adjacent to the vertex under consideration. By contrast, a 0 in the row indicates that the vertex in the column label is not adjacent to the vertex under consideration. From the standpoint of efficiency, *AdjacencyMat* is somewhat wasteful. The structure of the nested *for* loops alone cause the function to go cubic. In addition to this, the MATLAB functions that are called within *AdjacencyMat*, such as *strcat, num2str, and base2dec*, undoubtedly come at a cost as well.

Another matrix that can be used to represent a de Bruijn graph is a distance matrix. Unlike an adjacency matrix, a distance matrix shows the distance from every node to every other node in the graph. In our de Bruijn library of functions there is a method to generate a directed distance matrix. The code follows.

Algorithm 6 DirectedDistanceMat: Generates Directed Distance Matrix

```
1: procedure DirectedDistanceMat(d, n)
2:     N = d^n
3:     for i from 1 to N do
4:         for j from 1 to N do
5:             nodes(i, j) = DD((dec2base(i − 1, d, n)),(dec2base(j − 1, d, n)))
6:         end for
7:     end for
8: end procedure
9: return nodes
```

As you can see the function calls our library function DD(X, Y) in line 5, which computes the distance between two strings in the directed de Bruijn graph. Once the node names are generated, DD(X, Y) computes the distance and populates the *nodes* array. After exiting the outer *for* loop, *nodes* is returned. The cost of *DirectedDistanceMat* is at least quadratic in N, $O(N^2)$, due to the nested *for* loops, and this cost does not take into account the called library functions *dec2base()*. Since DD(d, n) is linear, it does not have much of an effect on efficiency. We executed DirectedDistance(2,4) using MATLAB which returned the following distance matrix in Table 4.2.

The undirected counterpart for *DirectedDistanceMat* in the DeBruijn library is *UndirectedDistanceMat*.

About the only thing worth mentioning for this function is that the time complexity and efficiency are very poor. This function calls UD(d, n) which

Table 1: Adjacency Matrix for $\vec{\mathcal{B}}(2, 4)$

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0001	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0010	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
0011	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0
0100	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0
0101	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	0
0110	0	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0
0111	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
1000	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1001	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0
1010	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	0
1011	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0
1100	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0
1101	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0
1110	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1111	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Table 2: Distance Matrix for $\vec{\mathcal{B}}(2, 4)$

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2
0001	2	0	2	2	1	1	1	1	2	2	2	2	2	2	2	2
0010	2	2	0	2	2	2	2	2	1	1	1	1	2	2	2	2
0011	2	2	2	0	2	2	2	2	2	2	2	2	1	1	1	1
0100	1	1	1	1	0	2	2	2	2	2	2	2	2	2	2	2
0101	2	2	2	2	1	0	1	1	2	2	2	2	2	2	2	2
0110	2	2	2	2	2	2	0	2	1	1	1	1	2	2	2	2
0111	2	2	2	2	2	2	2	0	2	2	2	2	1	1	1	1
1000	1	1	1	1	2	2	2	2	0	2	2	2	2	2	2	2
1001	2	2	2	2	1	1	1	1	2	0	2	2	2	2	2	2
1010	2	2	2	2	2	2	2	2	1	1	0	1	2	2	2	2
1011	2	2	2	2	2	2	2	2	2	2	2	0	1	1	1	1
1100	1	1	1	1	2	2	2	2	2	2	2	2	0	2	2	2
1101	2	2	2	2	1	1	1	1	2	2	2	2	2	0	2	2
1110	2	2	2	2	2	2	2	2	1	1	1	1	2	2	0	2
1111	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	0

Algorithm 7 UndirectedDistanceMat: Generates Undirected Dist. Mat.

```
1: procedure UndirectedDistanceMat(d, n)
2:     N = d^n
3:     for i from 1 to N do
4:         for j from 1 to N do
5:             nodes(i, j) = UD((dec2base(i − 1, d, n)),(dec2base(j − 1, d, n)))
6:         end for
7:     end for
8: end procedure
9: return nodes
```

runs at $O(n^3)$ efficiency. Exacerbating an already inefficient function, *UndirectedDistanceMat()* itself runs in $O(N^2)$ efficiency. This brings our efficiency down to $(O(n^3) \times O(N^2)) = O(N^3 d^{2n})$. The distance matrix representing the output for *UndirectedDistanceMat(2, 4)* is in Table 4.3.

As mentioned above, the reason *AdjacencyMat* (and the distance matrices) is so inefficient is because of its nested loop structure. Looking at the returned matrix in Table 4.1, one of the first things we notice is that the array is nearly filled with zeros. This is because de Bruijn graphs are sparsely populated with relatively few nodes when compared to their edges. The adjacency matrix goes through a lot of work to produce a relatively small amount of useful information. This next function, *VectorNeighborGenerator* returns the same useful information without the superfluous generation and storage of useless data. Note that this function omits the self-loops on nodes of the form a^n.

Running this method with the same parameters as *AdjacencyMat*$(2, 4)$ in the MATLAB command shell, we receive the following smaller data structure in return.

$$\text{VectorNeighborGenerator}(2, 4) :$$

$$\begin{aligned} \text{ans} \quad = \quad & \{0000 : 0001, 0001 : 0010, 0001 : 0011, 0010 : 0100, \\ & 0010 : 0101, 0011 : 0110, 0011 : 0111, 0100 : 1000, 0100 : 1001, \\ & 0101 : 1010, 0101 : 1011, 0110 : 1010, 1100 : 1101, 0111 : 1110, \\ & 0111 : 1111, 1000 : 0000, 1000 : 0001, 1001 : 0010, 1001 : 0011, \\ & 1010 : 0100, 1010 : 0101, 1011 : 0110, 1011 : 0111, 1100 : 1000, \\ & 1100 : 1001, 1101 : 1010, 1101 : 1011, 1110 : 1100, 1110 : 1101, \\ & 1111 : 1110\} \end{aligned}$$

This function is an improvement over *AdjacencyMat* because it runs in linear time complexity with respect to N (omitting any costs attributed from the called function *strcmp()*). Another improvement is that it does not return any useless values, and therefore instead of returning d^{2n} values it only returns $d^{n+1} - d$ values. In the case of $\vec{\mathcal{B}}(2, 4)$ above it returned 30 values instead of 256. This

Table 3: Distance Matrix for $\mathcal{B}(2,4)$

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	2	2	2	3	3	3	1	3	3	4	2	4	3	4
0001	1	0	1	1	2	2	2	2	1	2	3	3	2	3	3	3
0010	2	1	0	2	1	1	3	3	2	1	2	2	2	3	3	4
0011	2	1	2	0	2	3	1	1	2	1	3	2	2	2	2	2
0100	2	2	1	2	0	2	3	3	1	1	1	3	2	2	3	4
0101	3	2	1	3	2	0	2	2	3	2	1	1	3	2	3	3
0110	3	2	3	1	3	2	0	2	2	2	2	1	1	1	2	3
0111	3	2	3	1	3	2	2	0	3	2	3	1	2	2	1	1
1000	1	1	2	2	1	3	2	3	0	2	2	3	1	3	2	3
1001	3	2	1	1	1	2	2	2	2	0	2	3	1	3	2	3
1010	3	3	2	3	1	1	2	3	2	2	0	2	3	1	2	3
1011	4	3	2	2	3	1	1	1	3	3	2	0	2	1	2	2
1100	2	2	2	2	2	3	1	2	1	1	3	2	0	2	1	2
1101	4	3	3	2	2	2	1	2	3	3	1	1	2	0	1	2
1110	3	3	3	2	3	3	2	1	2	2	2	2	1	1	0	1
1111	4	3	4	2	4	3	3	1	3	3	3	2	2	2	1	0

Algorithm 8 VectorNeighborGenerator: Linked list rep. of graph

```
1: procedure VectorNeighborGenerator(d, n)
2:     N = d^n
3:     size = Nd − d
4:     nodes = GenerateNodes(d, n)
5:     edges = cell(1,size)
6:     nNeighbor = 0
7:     count = 1
8:     currentNode = 1
9:     for i from 1 to size+d do
10:        nNeighbor = nNeighbor + 1
11:        if nNeighbor > N then
12:            nNeighbor = 1
13:        end if
14:        if ∼strcmp(nodes{currentNode}, nodes{nNeighbor}) then
15:            edgescount – 1 =
16:                    strcat(nodes{currentNode}, char(':'), nodes{nNeighbor})
17:        else
18:            if count ≠ 1 then
19:                count = count – 1
20:            end if
21:        end if
22:        if (mod(i, d) = 0) and (nNeighbor ≠ 0) then
23:            currentNode = currentNode + 1
24:        end if
25:        count = count + 1
26:    end for
27: end procedure
28: return edges
```

is an 88.3% improvement in efficiency. Tracing the code, note that the variable *size* is instantiated to be $Nd - d$. This is the length of the return structure, a linear *cell array* called *edges*.

Also note that in line 4 *VectorNeighborGenerator* makes use of our library function *GenerateNodes*. It does this so that it may perform a string comparison rather than a nested loop search approach. The function uses a couple of variables *currentNode* to represent the current node under consideration and *nNeighbor*. The variable *nNeighbor* gets assigned by following a programmed route of nodes that are within the *currentNode's* "reach" when calculating its d –length *hops*. These *hops* follow a predictable path as illustrated below in Figure 11.

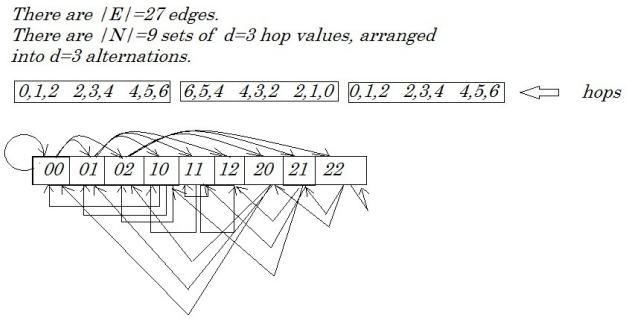


Figure 11: *Hops* in $\vec{\mathcal{B}}(3, 2)$

Using *currentNode* and *nNeighbor* in tandem, the linear array *nodes* is traversed, *currentNode* simply iterating in sequence while *nNeighbor* follows its *hopping* route. In line 14 a negated comparison is made to ensure that they are not equal, indicating a self–pointing node, and if this comparison fails then the two nodes are assigned together as one slot in the *edges* cell array separated by a colon (assignment done in line 15). After the loop counting variable reaches $i =$ size $+ d$ (in MATLAB array indices always start at 1) we exit the loop and return *edges*, the populated cell array.

Generating Balls and Spheres

The last set of functions involve the generation of two very similar yet distinctly different lists of nodes. The first is a function to generate a list of all nodes *within* a prescribed distance from a given node. The de Bruijn library functions for generating a set of all nodes contained in a t-ball follow.

An example of this function is given below.

DirectedBallFromX(1, 01, 3, 2) ans $= \{01, 10, 11, 12\}$

This function calls *DirectedDistanceMat* after which it parses the matrix looking for nodes within the specified distance, $t = 1$ from a specified node $X =$

Algorithm 9 DirectedBallFromX: Generates t-out ball from X

```
1: procedure DirectedBallFromX(t, X, d, n)
2:     N = d^n
3:     j = 1
4:     x = base2dec(X, d) + 1
5:     nodes = cell(1)
6:     B = DirectedDistanceMat(d, n)
7:     for i from 1 to N do
8:         if B(x, i) ≤ t then
9:             nodes{j} = dec2base(i − 1, d, n)
10:            j = j + 1
11:        end if
12:    end for
13: end procedure
14: return nodes
```

01 (line 8). When found, these nodes within t are converted to the desired base and placed into the cell array, *nodes* for return. The cost of *DirectedBallFromX* is quadratic with respect to N because *DirectedDistanceMat* is called within it.

The undirected counterpart function to *DirectedBallFromX* is called *UndirectedBallFromX*. This function generates a list of nodes that are within a specified distance from a specified node also, but the list returned has twice as many nodes in it since the graph is undirected. This function also calls its appropriate distance matrix, *UndirectedDistanceMat*, to assess the distances between nodes.

Algorithm 10 UndirectedBallFromX: Generates t-ball from X

```
1: procedure UndirectedBallFromX(t, X, d, n)
2:     N = d^n
3:     j = 1
4:     x = base2dec(X, d) + 1
5:     nodes = cell(1)
6:     B = UndirectedDistanceMat(d, n)
7:     for i from 1 to N do
8:         if B(x, i) ≤ t then
9:             nodes{j} = dec2base(i − 1, d, n)
10:            j = j + 1
11:        end if
12:    end for
13: end procedure
14: return nodes
```

As expected *UndirectedBallFromX* requires $O(N^3N^2)$ time efficiency because it calls *UndirectedDistanceMat* (which runs in $O(N^2)$ time). Although *UndirectedDistanceMat* works well, it is not at all efficient.

Let us examine an application of an undirected de Bruijn graph where the

function *UndirectedBallFromX* is utilized. Imagine a radio network modeled on the undirected de Bruijn graph $\mathcal{B}(2,4)$ where the repeater tower, represented by node *1100* is experiencing interference. In order for our transmitter, node *0000*, to reach our receiver, node *1111*, we must find a repeater tower path that avoids the defective tower. Executing the function *UndirectedBallFromX(1100, 1, 2, 4)* returns the list $\{0110, 1000, 1001, 1100, 1110\}$. By avoiding transmission through these towers, a fault free route can be determined. The shortest path is highlighted in green and it follows nodes $\{0001, 0011, 0111, 1111\}$ (see Figure 12).

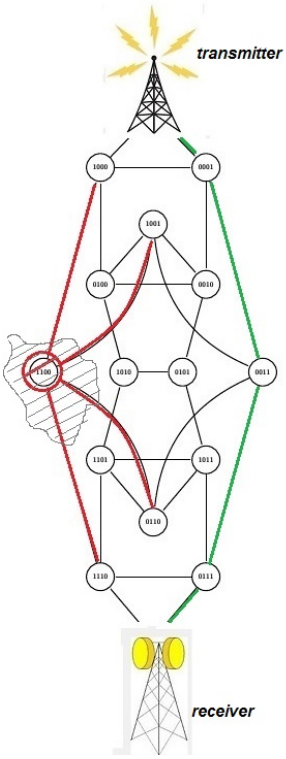


Figure 12: Radio network $\mathcal{B}(2,4)$ showing avoidance of fault paths

The function *DirectedAtFromX* is similar to *DirectedDFromX* except that instead of locating all nodes *within* a given distance, the function locates all nodes *at* the designated distance. The set of all nodes at exactly distance t from X is also known as a t-**sphere** centered at X.

An example of this function is given below.

DirectedAtFromX(1, 01, 3, 2) $ans = \{10, 11, 12\}$

This function calls *DirectedDistanceMat* after which it parses the matrix looking for nodes **at** the specified distance, $t = 1$ from a specified node *X=01* (line 8). When found, these nodes at t are converted to the desired base and

Algorithm 11 DirectedAtFromX: Generates t-sphere from X

```
1: procedure DirectedAtFromX(t, X, d, n)
2:     N = d^n
3:     j = 1
4:     x = base2dec(X, d) + 1
5:     nodes = cell(1)
6:     B = DirectedDistanceMat(d, n)
7:     for i from 1 to N do
8:         if B(x, i) = t then
9:             nodes{j} = dec2base(i − 1, d, n)
10:            j = j + 1
11:        end if
12:    end for
13: end procedure
14: return nodes
```

placed into the *cell array, nodes* for return. Like *DirectedBallFromX*, the cost of *DirectedAtFromX* is, quadratic time efficiency.

The undirected counterpart function to *DirectedAtFromX* is called *UndirectedAtFromX*. This function generates a list of nodes that are at a specified distance from a specified node also, but the list returned has more nodes in it since the graph is undirected. This function also calls its appropriate distance matrix, *UndirectedDistanceMat*, to assess the distances between nodes.

Returning to our radio tower scenario, where the tower configuration is modeled after the de Bruijn network $\mathcal{B}(2, 4)$. This time, imagine that the transmitter system underwent an upgrade and so now it is capable of propagating signals at much higher power. Our function *UndirectedAtFromX* could be utilized to determine broadcast range from the transmitter tower, node 0000. Let us say we broadcast a signal at four discrete and incrementally higher power levels, 1 - 4. After each transmission, we pause and await confirmation of receipt from the other towers in our network. After our first broadcast, at power level 1, we receive notification from towers 0001, and 1000. After our broadcast at power level 2 we receive confirmation from $0010, 0011, 0100, 1100$ as well as those who previously acknowledged. On power level number 3 we receive notification from towers $0101, 0110, 0111, 1001, 1010,$ and $, 1110$ as well as all those who have previously acknowledged. Finally we broadcast at power level 4, and we receive acknowledgement from towers $1011, 1101,$ and 1111 in addition to all other towers in the network (Refer to Figure 13).

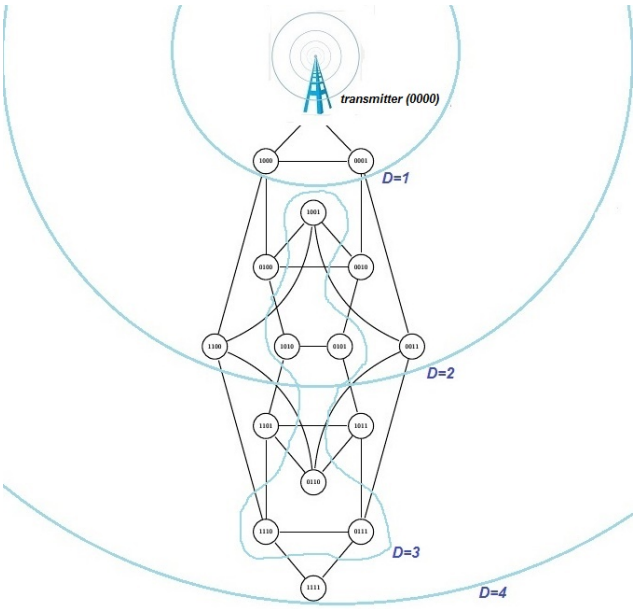


Figure 13: Radio Network $\mathcal{B}(2,4)$ showing Broadcast Distance

4.5 Recursive Constructions

The standard, well-known recursive construction for the de Bruijn graph inducts on the string length, n, while holding the alphabet size d fixed. This construction illustrates the fact that $\vec{\mathcal{B}}(d,n)$ is the line graph of $\vec{\mathcal{B}}(d,n-1)$. For a good description, see [2].

Instead, we will focus on a new construction that increases the alphabet size while holding the string length fixed. Our construction is as follows.

Construction 4.91. *To construct $\mathcal{B}(d,n)$ from $\mathcal{B}(d-1,n)$, we do the following.*

1. *Make $d+1$ copies of $\mathcal{B}(d-1,n)$ labeled "Copy i" for $i \in \{0,1,2,\ldots,d\}$.*
2. *For each $i \in \{0,1,2,\ldots,d-1\}$, in Copy i we replace every occurrence of letter i with letter d. Leave Copy d unaltered, i.e. Copy $d = \mathcal{B}(d-1,n)$.*
3. *Combine all of the new copies to obtain $\mathcal{B}(d,n)$ as follows.*
   (a) *New vertices: Any vertex containing the letter d, i.e. all strings of length n with at least one d.*
   (b) *New edges: New edges come from the copies as follows:*
   Copy 0: *All edges containing d*
   Copy 1: *All edges containing $0,d$*
   Copy 2: *All edges containing $0,1,d$*

Copy 3: *All edges containing* $0, 1, 2, d$

$\vdots$

Copy k**:** *All edges containing* $0, 1, 2, \ldots, k-1, d$

$\vdots$

Copy $d-1$**:** *All edges containing* $0, 1, 2, \ldots, d-1, d$

We now illustrate this construction to obtain $\mathcal{B}(4,2)$ from $\mathcal{B}(3,2)$. New edges are colored. For each copy, we count how many new edges are added to ensure that we have a total of $4^3 = 64$ edges represented for $\mathcal{B}(4,2)$. We will use the principle of inclusion-exclusion to count the number of edges added.

Example:

Copy 3: Note that Copy 3 $= \mathcal{B}(3,2)$.

Copy 3	00	01	02	10	11	12	20	21	22
00	2	1	1	1	0	0	1	0	0
01	1	0	0	2	1	1	1	0	0
02	1	0	0	1	0	0	2	1	1
10	1	2	1	0	1	0	0	1	0
11	0	1	0	1	2	1	0	1	0
12	0	1	0	0	1	0	1	2	1
20	1	1	2	0	0	1	0	0	1
21	0	0	1	1	1	2	0	0	1
22	0	0	1	0	0	1	1	1	2

New edges: $3^3 = 27$.

Copy 0:

Copy 0	33	31	32	13	11	12	23	21	22
33	2	1	1	1	0	0	1	0	0
31	1	0	0	2	1	1	1	0	0
32	1	0	0	1	0	0	2	1	1
13	1	2	1	0	1	0	0	1	0
11	0	1	0	1	2	1	0	1	0
12	0	1	0	0	1	0	1	2	1
23	1	1	2	0	0	1	0	0	1
21	0	0	1	1	1	2	0	0	1
22	0	0	1	0	0	1	1	1	2

New edges:

Total edges:	$+3^3$	27
- edges w/o 3:	-2^3	-8
		19

Copy 1:

Copy 1	00	03	02	30	33	32	20	23	22
00	2	1	1	1	0	0	1	0	0
03	1	0	0	2	1	1	1	0	0
02	1	0	0	1	0	0	2	1	1
30	1	2	1	0	1	0	0	1	0
33	0	1	0	1	2	1	0	1	0
32	0	1	0	0	1	0	1	2	1
20	1	1	2	0	0	1	0	0	1
23	0	0	1	1	1	2	0	0	1
22	0	0	1	0	0	1	1	1	2

New edges:

Total edges:	$+3^3$	27
- edges w/o 0:	-2^3	-8
- edges w/o 3:	-2^3	-8
+ edges w/o 0,3:	$+1^3$	$+1$
		12

Copy 2:

Copy 2	00	01	03	10	11	13	30	31	33
00	2	1	1	1	0	0	1	0	0
01	1	0	0	2	1	1	1	0	0
03	1	0	0	1	0	0	2	1	1
10	1	2	1	0	1	0	0	1	0
11	0	1	0	1	2	1	0	1	0
13	0	1	0	0	1	0	1	2	1
30	1	1	2	0	0	1	0	0	1
31	0	0	1	1	1	2	0	0	1
33	0	0	1	0	0	1	1	1	2

New edges:

Total edges:	$+3^3$	27
- edges w/o 0:	-2^3	-8
- edges w/o 1:	-2^3	-8
- edges w/o 3:	-2^3	-8
+ edges w/o 0,1:	$+1^3$	$+1$
+ edges w/o 0,3:	$+1^3$	$+1$
+ edges w/o 1,3:	$+1^3$	$+1$
- edges w/o 0,1,3:	-0^3	-0
		6

This gives us a total of number edges: $27 + 19 + 12 + 6 = 64$, as desired.

Algorithm

We now wish to convert this process to an algorithm for a program such as Matlab to generate $\mathcal{B}(d+1,n)$ from $\mathcal{B}(d,n)$. From our previous discussions and examples, we know that we have the following new edges for each copy. We will rename Copy d to Copy -1, and our motivation for this will be clear in the subsequent counting formulas.

Copy -1: All $(n+1)$-strings on alphabet $\{0,1,2,\ldots,d-1\}$.

$\rightarrow$ Add d^{n+1} edges.

Copy 0: All $(n+1)$-strings on alphabet $\{0,1,2,\ldots,d-1,d\}\setminus\{0\}$ that contain the letter d.

$\rightarrow$ Add $d^{n+1}-\binom{1}{1}(d-1)^{n+1}$ edges.

Copy 1: All $(n+1)$-strings on alphabet $\{0,1,2,\ldots,d-1,d\}\setminus\{1\}$ that contain the letters $0,d$.

$\rightarrow$ Add $d^{n+1}-\binom{2}{1}(d-1)^{n+1}+\binom{2}{2}(d-2)^{n+1}$ edges.

Copy 2: All $(n+1)$-strings on alphabet $\{0,1,2,\ldots,d-1,d\}\setminus\{2\}$ that contain the letters $0,1,d$.

$\rightarrow$ Add $d^{n+1}-\binom{3}{1}(d-1)^{n+1}+\binom{3}{2}(d-2)^{n+1}-\binom{3}{3}(d-3)^{n+1}$ edges.

$\vdots$

Copy k: All $(n+1)$-strings on alphabet $\{0,1,2,\ldots,d-1,d\}\setminus\{k\}$ that contain the letters $0,1,2,\ldots,k-1,d$.

$\rightarrow$ Add $\sum_{i=0}^{k+1}(-1)^i\binom{k+1}{i}(d-i)^{n+1}$ edges.

$\vdots$

Copy $d-1$: All $(n+1)$-strings on alphabet $\{0,1,2\ldots,d-1,d\}\setminus\{d-1\}$ that contain the letters $0,1,2,\ldots,d-2,d$.

$\rightarrow$ Add $\sum_{i=0}^{d}(-1)^i\binom{d}{i}(d-i)^{n+1}$ edges.

This gives us that the total number of edges in $\mathcal{B}(d+1,n)$ is:

$$\begin{aligned}
\|\mathcal{B}(d+1,n)\| &= \sum_{k=-1}^{d-1}\|\text{Copy } k\| \\
&= \sum_{k=-1}^{d-1}\sum_{i=0}^{k+1}(-1)^i\binom{k+1}{i}(d-i)^{n+1} \\
&= \sum_{j=0}^{d}\sum_{i=0}^{j}(-1)^i\binom{j}{i}(d-i)^{n+1}
\end{aligned}$$

Note that this algorithm will never count strings that contain at least one of every letter in the set $\{0, 1, 2, \ldots, d\}$, and hence we must require that $n < d$. So we have now developed the following combinatorial identity.

Theorem 4.92. *For $n < d$,*

$$(d+1)^{n+1} = \sum_{j=0}^{d} \sum_{i=0}^{j} (-1)^i \binom{j}{i} (d-i)^{n+1}.$$

To address the cases where $n \geq d$, we must count the number of $(n+1)$-strings on $\{0, 1, 2, \ldots, d\}$ in which each letter appears at least once. This is equivalent to the number of onto functions from an $(n+1)$-set to a $(d+1)$-set. It is a well-known result and a standard example of the Principle of Inclusion-Exclusion (see [19] for a discussion of this principle) that this number of onto functions is given by:

$$\sum_{i=0}^{d+1} (-1)^i \binom{d+1}{i} (d+1-i)^{n+1}.$$

Thus we obtain the following result that covers all cases.

Theorem 4.93. *For all $n, d \in \mathbb{Z}^+$:*

$$\begin{aligned}(d+1)^{n+1} &= \left(\sum_{j=0}^{d} \sum_{i=0}^{j} (-1)^i \binom{j}{i} (d-i)^{n+1} \right) \\ &\quad + \left(\sum_{i=0}^{d+1} (-1)^i \binom{d+1}{i} (d+1-i)^{n+1} \right).\end{aligned}$$

Example

We now run through a complete example to construct $\mathcal{B}(3, 2)$ from $\mathcal{B}(2, 2)$.

Copy -1: We begin with the graph $\mathcal{B}(2, 2)$. See Figure 14.

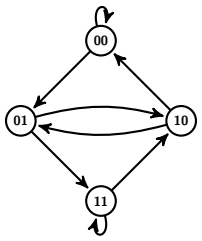


Figure 14: Copy -1

Copy 0: We add in a copy of $\mathcal{B}(2,2)$ where the letter 0 is renamed 2. See Figure 15 for Copy 0 and Figure 16 for the merged figure, with new edges in red.

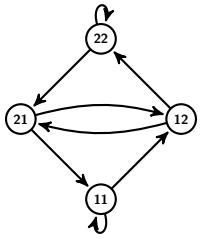


Figure 15: Copy 0

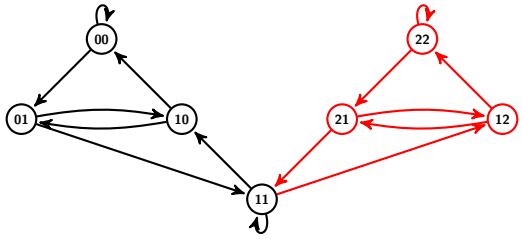


Figure 16: Merged Copy -1 and Copy 0

Copy 1: We add in a copy of $\mathcal{B}(2,2)$ where the letter 1 is renamed 2. See Figure 17 with new edges in blue.

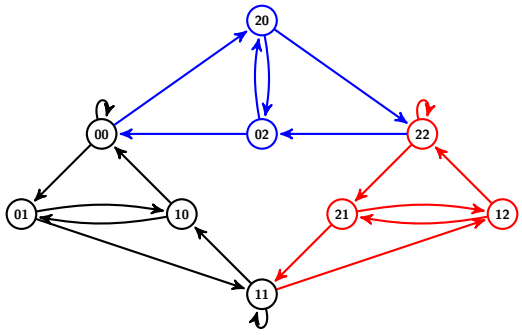


Figure 17: Merged Copy -1, Copy 0, and Copy 1

Additional: Since $n \geq d$, we must additionally consider all 3-strings that contain every letter from $\{0, 1, 2\}$ at least once. Note that this is the set of all permutations of $\{0, 1, 2\}$, so we add in the following edges.

$$\{012, 021, 102, 120, 201, 210\}$$

See Figure 18 with new edges in green.

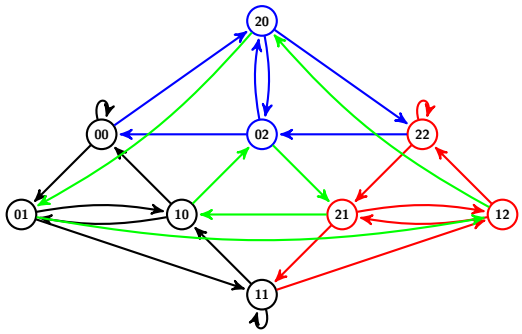


Figure 18: Merged Copy -1, Copy 0, Copy 1, and additional edges

4.6 Identifying Code Problem Formulations

As an NP-complete problem, computing base cases for the minimum identifying code problem is quite challenging. In this chapter, we explore various methods utilized to compute base cases around which to develop conjectures.

4.6.1 Parallel Computing

The pseudocode in Algorithm 12 describes our brute force algorithm, implemented in Matlab using the Parallel Computing Toolbox.

Algorithm 12 Brute Force Algorithm

```
1:  procedure BruteForce(d, n, k)          ▷ B(d, n) and subset size k
2:      Create list of all subsets of k nodes
3:      for i = 1 : (d^n choose k) do
4:          if subset i is a valid identifying code then
5:              Display subset i to user
6:          else
7:              Do nothing
8:          end if
9:      end for
10: end procedure
```

Parallelizing our algorithm takes two steps. The first step is to replace "For" on line 3 with "Parfor". This indicates to Matlab to use the parallel computing toolbox and run each loop iteration independently. The second step requires moving the construction of subsets inside the Parfor loop. Because of the exponential increase in the number of subsets created, it is more efficient to generate each subset within the loop and discard it after the iteration than to store all $\binom{d^n}{k}$ k-subsets and traverse through the list. This is done using a k-subset unranking algorithm. Two of these algorithms (from [16]) are listed as Algorithms 13 and 14. These unranking functions allow us to completely parallelize the brute force algorithm, and the results obtained are listed in Figure 19.

Algorithm 13 Revolving Door Unranking Algorithm

1: **procedure** RevDoor(r, k, n) ▷ subset index, subset size, set size
2: $x = n$
3: **for** $i = k : 1$ **do**
4: **while** $\binom{x}{i} > r$ **do**
5: $x = x - 1$
6: **end while**
7: $t_i = x + 1$
8: $r = \binom{x+1}{i} - r - 1$
9: **end for**
10: **end procedure**
11: **return** $T = (t_1, t_2, \ldots, t_k)$

Algorithm 14 Lexicographic Unranking Algorithm

1: **procedure** LexUnrank(r, k, n) ▷ subset index, subset size, set size
2: $x = 1$
3: **for** $i = 1 : k$ **do**
4: **while** $r \geq \binom{n-x}{k-i}$ **do**
5: $r = r - \binom{n-x}{k-i}$
6: $x = x + 1$
7: **end while**
8: $t_i = x$
9: $x = x + 1$
10: **end for**
11: **end procedure**
12: **return** $T = (t_1, t_2, \ldots, t_k)$

4.6.2 D-Wave Quantum Annealing Machine

Under the collaborative effort "Adiabatic Quantum Computing Applications Research" (14-RI-CRADA-02) between the Information Directorate and Lock-

$d \setminus n$	2	3	4	5
2	$\times$	4	6	12
3	4	9		
4	5			
5	6			

Figure 19: Results for $\mathcal{B}(d, n)$ obtained using HPC

heed Martin Corporation, we aim to extend the results obtained by the parallel computing method. In general, the D-Wave machine can address a class of Ising problems natively by the hardware. As stated in the D-Wave user documents, "The D-Wave hardware can be viewed as a hardware heuristic which minimizes Ising objective functions using a physically realized version of quantum annealing." [9] The Ising model is an energy minimization problem of -1/+1-valued variables. It can be converted to a quadratic unconstrained binary optimization (QUBO) problem that uses 0/1-valued variables, and so they are often used interchangeably.

Binary Optimization Model

We present a binary optimization formula for the 1-identifying code problem. Adjustments must be made to create a quadratic version. We will define this model using three separate functions: one to show that the set has the correct size, one to show that the set is dominating, and one to show that the set is separating (or identifying).

Variable Definitions

We will use the notation $B(v)$ for $v \in V(G)$, where $B(v) = N(v) \cup \{v\}$. In other words, $B(v)$ is the set containing all vertices adjacent to v, plus v itself.

We define the variables as follows.

$$x_{vi} = \begin{cases} 1, & \text{if } i \in B(v); \\ 0, & \text{otherwise.} \end{cases}$$

Set S has size k

We define the first function, H_A, as follows.

$$\begin{aligned} H_A &= (k - \textstyle\sum_v x_{vv}) \\ &= 0 \qquad \text{iff} \quad |S| = k. \end{aligned}$$

Set S is a dominating set

By definition, this is equal to $\forall v \in G,\ B(v) \cap S \neq \emptyset$ This is equivalent to the following.

$$\begin{aligned}
\forall v \in G, B(v) \cap S \neq \emptyset \quad &\leftrightarrow \quad (x_{vv} = 1) \vee \left(\sum_{uv \in E} x_{uv} \geq 1 \right) \\
&\leftrightarrow \quad (1 - x_{vv} = 0) \vee \neg \left(\sum_{uv \in E} x_{uv} = 0 \right) \\
&\leftrightarrow \quad (1 - x_{vv} = 0) \vee \left(\prod_{uv \in E} (1 - x_{uv}) = 0 \right)
\end{aligned}$$

From this statement, we get the following equation for our second function.

$$H_B \quad = \quad \textstyle\sum_v (1 - x_{vv}) \cdot \left(\prod_{uv \in E} (1 - x_{uv}) \right)$$

Set S is a separating set

By definition, this is equal to $\forall x, y \in G,\ (B(x) \cap S) \triangle (B(y) \cap S) \neq \emptyset$. This is equivalent to the following for a specific pair $x \neq y$.

$$\begin{aligned}
(B(x) \cap S) \triangle (B(y) \cap S) \neq \emptyset \quad &\leftrightarrow \quad \exists v \in (B(x) \cap S) \triangle (B(y) \cap S) \\
&\leftrightarrow \quad \exists v, (v \in B(x) \cap S) \oplus (v \in B(y) \cap S) \\
&\leftrightarrow \quad \exists v, (x_{xv} = 1) \oplus (x_{yv} = 1) \\
&\leftrightarrow \quad \exists v, (1 - (x_{xv} + x_{yv}) = 0) \\
&\leftrightarrow \quad \prod_v (1 - x_{xv} - x_{yv}) = 0
\end{aligned}$$

From this statement, we get the following equation for our third function, summed over all pairs x, y.

$$H_C \quad = \quad \textstyle\sum_x \sum_{y \neq x} \prod_v (1 - x_{xv} - x_{yv})^2$$

The Binary Optimization Model

From these three functions, our binary optimization model is the following.

$$\begin{aligned}
H(S) \quad &= \quad H_A(S) + H_B(S) + H_C(S) \\
&= \quad 0 \qquad\qquad \text{iff} \quad \text{S is an identifying code.}
\end{aligned}$$

Note that while this does provide a binary optimization model for our problem, it is not quadratic. In order to convert $H(S)$ to a quadratic binary equation, each higher order term must be replaced with several new variables. While this is possible, it is a time-consuming and arduous process that introduces many new variables. Hence this approach will likely not be the most efficient implementation.

Integer Program Formulation

From [20], we have the following integer program formulation of the minimum identifying code problem in directed graphs.

First, we define the **modified adjacency matrix** as follows. It is the adjacency matrix plus the identity matrix.

$$A_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \text{ or } i = j; \\ 0, & \text{otherwise.} \end{cases}$$

Using this definition, we see that a ball of radius 1 surrounding vertex i is given by the following vector.

$$B(i) = [A_{1i}, A_{2i}, \ldots, A_{ni}]^T$$

Our vertex subset S is defined as the following vector.

$$S = [s_1, s_2, \ldots, s_n]^T \text{ where } s_i = \begin{cases} 1, & \text{if } i \in S; \\ 0, & \text{otherwise.} \end{cases}$$

To compare two identifying sets with respect to S for vertices i and j, the following expression computes the size of $(B(i) \cap S) \triangle (B(j) \cap S)$.

$$\sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k$$

This implies that in order for S to be a valid 1-identifying code, we must have the following inequality satisfied for all pairs of vertices i and j.

$$\sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k \geq 1$$

For the dominating property to be satisfied, we require the following additional inequality.

$$A \cdot S \geq \mathbf{1}^T$$

Thus our integer program is given by the following.

$$\begin{array}{lllll} \min & |S| & & & \\ \text{s.t.} & \sum_{k=1}^{n} |A_{ki} - A_{kj}| \cdot s_k & \geq & 1, & \forall i \neq j \\ & A \cdot S & \geq & \mathbf{1}^T & \\ & s_k \in \{0,1\} & & & \end{array}$$

In order to use these ideas for the D-Wave machine, our constraints must be equalities. This means we must add binary slack variables for each inequality. For the first set of inequalities, we must determine an upper bound for each inequality. Since these correspond to the constraint $|(B(i) \cap S) \triangle (B(j) \cap S)| \geq 1$, an easy upper bound is given by the following.

$$|B(i)| + |B(j)| \geq |(B(i) \cap S) \triangle (B(j) \cap S)| \geq 1$$

For the class of de Bruijn graphs, we are able to use this to get a bound on the number of slack variables needed. Since the maximum size of any ball in $\mathcal{B}(d,n)$ is $2d+1$, this gives us an upper bound of size $4d+2$ for this class of graphs. Hence for each inequality in this set, we must add $4d+2$ binary slack variables to convert the inequality to an equality. Since there are $\frac{d^n(d^n-1)}{2}$ possible pairs i,j, this implies that we must add a huge number of binary slack variables, equal to the following expression, just to satisfy the first set of inequalities.

$$d^n(d^n-1)(2d+1) \text{ slack variables}$$

Hence, this method is not going to be an efficient way to map our problem onto the D-Wave machine.

Satisfiability Formulation

This approach formulates the identifying code problem as a boolean satisfiability problem. Each term in the satisfiability problem is mapped to an Ising model. The mapping introduces auxiliary variables so that the Ising model contains only quadratic and linear terms. A graph embedding technique is then used to map this Ising model onto the connectivity of the D-Wave chip. Finally, gauge transformations are used to mitigate the effects of intrinsic control errors. We will illustrate each step with an example of $\mathcal{B}(d,n)$ when $d=2$ and $n=3$. As stated previously, these methods easily apply to any graph.

Satisfiability Formulation

First, we label the nodes of $\mathcal{B}(2,3)$ from 0 to $d^n-1=7$. Then we define the set of variables $\{x_i\}$ for $i=0,1,\ldots,7$ as follows.

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is included in the identifying code;} \\ 0, & \text{otherwise.} \end{cases}$$

Next, we look at the ball for each node and form clauses corresponding to their domination constraints.

Ball	Contents	Constraints
$B(0)=B(000)$	$\{000,001,100\}=\{0,1,4\}$	$x_0 \vee x_1 \vee x_4$
$B(1)=B(001)$	$\{000,001,010,011,100\}=\{0,1,2,3,4\}$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee v_4$
$B(2)=B(010)$	$\{001,010,100,101\}=\{1,2,4,5\}$	$x_1 \vee x_2 \vee x_4 \vee x_5$
$B(3)=B(011)$	$\{001,011,101,110,111\}=\{1,3,5,6,7\}$	$x_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$
$B(4)=B(100)$	$\{000,001,010,100,110\}=\{0,1,2,4,6\}$	$x_0 \vee x_1 \vee x_2 \vee x_4 \vee x_6$
$B(5)=B(101)$	$\{010,011,101,110\}=\{2,3,5,6\}$	$x_2 \vee x_3 \vee x_5 \vee x_6$
$B(6)=B(110)$	$\{011,100,101,110,111\}=\{3,4,5,6,7\}$	$x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$
$B(7)=B(111)$	$\{011,110,111\}=\{3,6,7\}$	$x_3 \vee x_6 \vee x_7$

From this set of constraints, we form clauses for each pairwise XOR of balls. This is shown in Figure 20

Now we can eliminate more specific clauses that are implied by more general clauses. For example, Figure 21 shows which two-term constraints imply the corresponding larger constraints. Hence, the only constraints that we have left are given below.

	$B(1)$	$B(2)$	$B(3)$	$B(4)$	$B(5)$	$B(6)$	$B(7)$
$B(0)$	$x_2 \vee x_3$	$x_0 \vee x_2 \vee x_5$	$x_0 \vee x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$	$x_2 \vee x_6$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6$	$x_0 \vee x_1 \vee x_3 \vee x_5 \vee x_6 \vee x_7$	$x_0 \vee x_1 \vee x_3 \vee x_4 \vee x_6 \vee x_7$
$B(1)$		$x_0 \vee x_3 \vee x_5$	$x_0 \vee x_2 \vee x_4 \vee x_5 \vee x_6 \vee x_7$	$x_3 \vee x_6$	$x_0 \vee x_1 \vee x_4 \vee x_5 \vee x_6$	$x_0 \vee x_1 \vee x_2 \vee x_5 \vee x_6 \vee x_7$	$x_0 \vee x_1 \vee x_2 \vee x_4 \vee x_6 \vee x_7$
$B(2)$			$x_2 \vee x_3 \vee x_4 \vee x_6 \vee x_7$	$x_0 \vee x_5 \vee x_6$	$x_1 \vee x_3 \vee x_4 \vee x_6$	$x_1 \vee x_2 \vee x_3 \vee x_6 \vee x_7$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_6 \vee x_7$
$B(3)$				$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5 \vee x_7$	$x_1 \vee x_2 \vee x_7$	$x_1 \vee x_4$	$x_1 \vee x_5$
$B(4)$					$x_0 \vee x_1 \vee x_3 \vee x_4 \vee x_5$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_5 \vee x_7$	$x_0 \vee x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_7$
$B(5)$						$x_2 \vee x_4 \vee x_7$	$x_2 \vee x_5 \vee x_7$
$B(6)$							$x_4 \vee x_5$

Figure 20: XOR balls for case $d = 2$, $n = 3$

General Constraint	Specific Constraints
$B(0) \oplus B(1) = x_2 \vee x_3$	$B(0) \oplus B(5), B(2) \oplus B(3), B(2) \oplus B(6), B(3) \oplus B(4), B(4) \oplus B(6), B(4) \oplus B(7)$
$B(0) \oplus B(4) = x_2 \vee x_6$	$B(0) \oplus B(5), B(1) \oplus B(3), B(1) \oplus B(6), B(1) \oplus B(7), B(2) \oplus B(3), B(2) \oplus B(6), B(2) \oplus B(7)$
$B(1) \oplus B(4) = x_3 \vee x_6$	$B(0) \oplus B(3), B(0) \oplus B(5), B(0) \oplus B(6), B(0) \oplus B(7), B(2) \oplus B(3), B(2) \oplus B(5), B(2) \oplus B(6), B(2) \oplus B(7)$
$B(3) \oplus B(6) = x_1 \vee x_4$	$B(0) \oplus B(5), B(0) \oplus B(7), B(1) \oplus B(5), B(1) \oplus B(7), B(2) \oplus B(5), B(2) \oplus B(7), B(3) \oplus B(4), B(4) \oplus B(5), B(4) \oplus B(7)$
$B(3) \oplus B(7) = x_1 \vee x_5$	$B(0) \oplus B(5), B(0) \oplus B(6), B(1) \oplus B(2), B(1) \oplus B(5), B(1) \oplus B(6), B(2) \oplus B(7), B(3) \oplus B(4), B(4) \oplus B(5), B(4) \oplus B(6)$
$B(6) \oplus B(7) = x_4 \vee x_5$	$B(0) \oplus B(3), B(0) \oplus B(5), B(1) \oplus B(3), B(1) \oplus B(5), B(2) \oplus B(7), B(3) \oplus B(4), B(4) \oplus B(5)$

Figure 21: Constraint Implications

$$
\begin{aligned}
&x_2 \vee x_3 \\
&x_0 \vee x_2 \vee x_5 \\
&x_2 \vee x_6 \\
&x_0 \vee x_3 \vee x_5 \\
&x_3 \vee x_6 \\
&x_0 \vee x_5 \vee x_6 \\
&x_1 \vee x_2 \vee x_7 \\
&x_1 \vee x_4 \\
&x_1 \vee x_5 \\
&x_2 \vee x_4 \vee x_7 \\
&x_2 \vee x_5 \vee x_7 \\
&x_4 \vee x_5
\end{aligned}
$$

Satisfying this set of constraints are the four possible minimum solutions, given below.

$$
\begin{aligned}
&\{x_1, x_2, x_3, x_5\} \\
&\{x_1, x_2, x_5, x_6\} \\
&\{x_2, x_3, x_4, x_5\} \\
&\{x_2, x_4, x_5, x_6\}
\end{aligned}
$$

Mapping Satisfiability Clauses to Ising Models

We construct a Hamiltonian

$$\mathcal{H} = \sum_j \mathcal{H}_j(\{x_i, i \in A_j\}) + \lambda \sum_i x_i.$$

Each of the terms has the property that

$$x^* = \arg\min_{x_i} \mathcal{H}_j(\{x_i, i \in A_j\}) \text{ iff } \left(\bigvee_{i \in A_j} x_i \text{ is true} \right).$$

We will show momentarily how the $\mathcal{H}_j$ are constructed. The last term $\lambda > 0$ is a penalty term that rewards shorter length codes. Therefore, the minimum solutions (or ground states) of $\mathcal{H}$ are the minimum 1-identifying codes.

In order to solve the Hamiltonian using adiabatic quantum optimization, we have the further constraint that the $\mathcal{H}_j$ must contain only quadratic and linear terms in the binary variables $\{x_i\}$. In general to accomplish this, we must introduce auxiliary variables, which we will denote by $\{z_i\}$. Also, we will switch to the Ising model convention where each of the x_i and z_i can take values $\{+1, -1\}$ instead of $\{0, 1\}$.

The mapping from OR-clauses to Ising models that we will use, namely

$$\bigvee_{i \in A_j} x_i \mapsto \mathcal{H}_j(\{x_i, i \in A\}),$$

depends only on the number of variables $k = |A_j|$ in the OR-clause. These mappings for $k = 2$ through $k = 6$ are represented diagrammatically in Figure 22.

In the diagrams, numbers attached to a node represent the linear coefficients in the Ising model, while numbers attached to an edge represent the quadratic (coupling) coefficients in the Ising model. For example, the diagram for $k = 3$ represents the following Ising model.

$$\mathcal{H}_3(x_1, x_2, x_3, z_1) = x_1x_2 - 2x_1z_1 - 2x_2z_1 - 2x_2z_1 + z_1x_3 + x_1 + x_2 - 3z_1 - x_3$$

It can be confirmed that every ground state of $\mathcal{H}_3(x_1, x_2, x_3, z_1)$ satisfies $x_1 \vee x_2 \vee x_3$, and conversely every combination of $\{x_1, x_2, x_3\}$ that satisfies $x_1 \vee x_2 \vee x_3$ corresponds to a ground state of $\mathcal{H}_3(x_1, x_2, x_3, z_1)$.

Mapping the Ising model onto the D-Wave processor

In general, the graph of the Hamiltonian we get from the satisfiability-to-Ising mapping will *not* fit onto the D-Wave hardware graph. The D-Wave hardware graph, which is called the "Chimera" graph, is built up of unit cells, each of which is a four by four bipartite graph, $\mathcal{K}_{4,4}$. Even the simple Ising model for 3-OR shown in Figure 22 cannot be mapped directly onto the D-Wave hardware graph. This can be seen from the fact that our graph $\mathcal{B}(2, 3)$ contains a 3-cycle, whereas the smallest cycle possible on the D-Wave hardware graph is a 4-cycle.

Embedding

Our first step to embedding is to determine how to map our OR-clauses to the physical qubits. One way to embed the 3-OR graph onto the D-Wave is shown in Figure 23. We have mapped the logical qubit z_1 to two physical qubits, which are ferromagnetically coupled with a coupling strength -JFm.

Similarly, once we constructed the full Ising Hamiltonian for the minimum 1-independent code problem, we can use embedding to map the graph of the Hamiltonian onto the D-Wave hardware graph.

Problem Decomposition

If the graph of the Hamiltonian is too large to embed onto our current 504-qubit D-Wave hardware graph, one trick we can try is to decompose the satisfiability problem into smaller pieces that can be embedded. For example, in the $\mathcal{B}(2, 3)$ example from earlier, one of the terms is $x_2 \vee x_3$. In order for this to be true, at least one of x_2 and x_3 must be true. We consider and solve each case separately.

If x_2 is true, then so is any OR-clause containing x_2, so we can eliminate

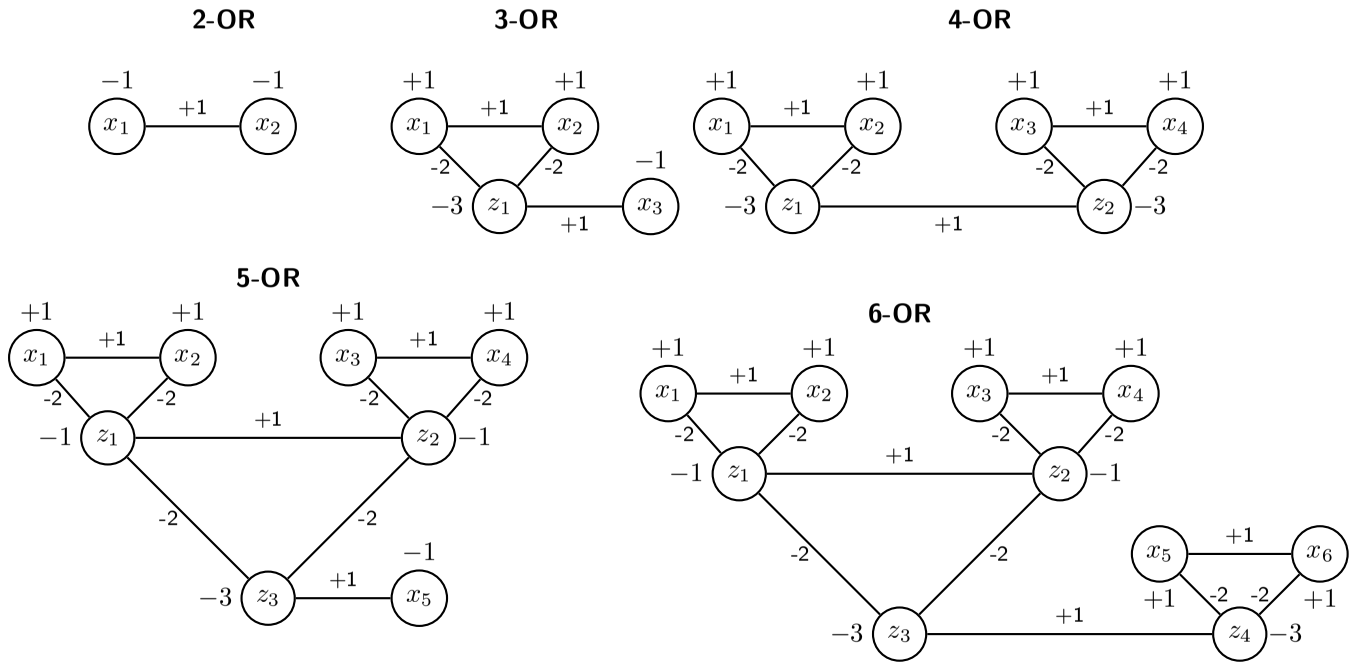


Figure 22: Mapping from OR-clauses to Ising Models

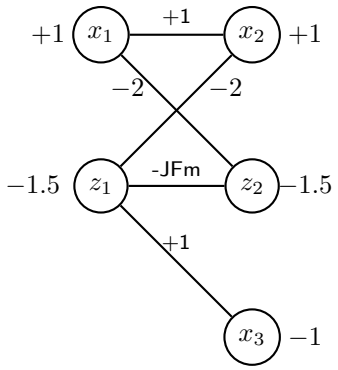


Figure 23: Embedding 3-OR onto the Chimera Graph

those from the conjunction. We are left with the following clauses.

$$x_3 \vee x_6$$
$$x_0 \vee x_5 \vee x_6$$
$$x_1 \vee x_4$$
$$x_1 \vee x_5$$
$$x_4 \vee x_5$$

Similarly, in the case where x_3 is true we can eliminate terms containing x_3, leaving the following clauses.

$$x_2 \vee x_6$$
$$x_0 \vee x_5 \vee x_6$$
$$x_1 \vee x_2 \vee x_7$$
$$x_1 \vee x_4$$
$$x_1 \vee x_5$$
$$x_2 \vee x_4 \vee x_7$$
$$x_2 \vee x_5 \vee x_7$$
$$x_4 \vee x_5$$

Both of these subproblems are simpler than the original problem and hence easier to embed. Whichever subproblem yields the smaller identifying codes will be the solution of our original problem, or if both subproblems have minimum solutions of the same length, then we can take the union of the two solution sets.

Gauge Transformations

Embedding complex graphs leads to long *chains*, i.e. multiple physical qubits corresponding to the same logical qubit. In the current D-Wave embedding solver, all of the physical qubits in a chain will be ferromagnetically coupled

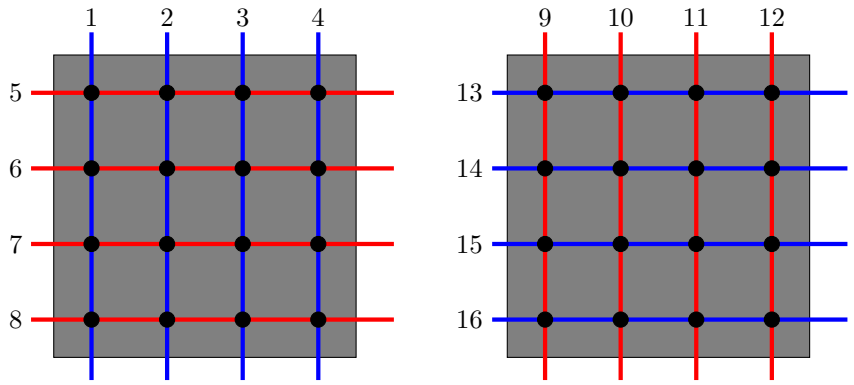


Figure 24: Example Gauge Transformation

with some coupling strength -JFm, which is iteratively increased to a large enough magnitude to ensure that all of the physical qubits in the chain agree (most of the time) and can be treated as a single logical qubit.

However, due to the characteristics of the D-Wave design, the control precision for implementing ferromagnetic couplings is somewhat worse than for antiferromagnetic couplings. So, embeddings with many long chains will have a greater tendency for control precision errors which may affect solution quality. To combat these effects, we can utilize a *gauge transformation*, where we redefine a subset of the spin variables to be the opposite sign. Flipping a subset of the spins in this way induces a transformation on the Ising coefficients: If S_1 has been flipped ($S_1' = -S_1$), then $h_1' = -h_1$. If one of S_1 and S_2 has been flipped, then $J_{12}' = -J_{12}$. But if both of S_1 and S_2 have been flipped, then $J_{12}' = J_{12}$.

Consider for example the gauge transformation shown in Figure 24. In the figure, the red qubits are flipped by the gauge transformation while the blue qubits are unchanged. In the first unit cell, all of the horizontal qubits are flipped while the vertical qubits are unchanged. In the next unit cell, all of the vertical qubits are flipped while the horizontal qubits are unchanged. So, suppose our embedding contains the chain 1 (blue vertical qubit), 5 (red horizontal qubit), 13 (blue horizontal qubit). Note that each consecutive pair in the chain contains exactly one flipped qubit, so all of the ferromagnetic couplings -JFm in the chain will be replaced with antiferromagnetic couplings +JFm by the gauge transformation. By using gauge transformations like this, we may be able to reduce the control precision errors caused by embeddings with long chains.

Results for $\mathcal{B}(2,4)$

By combining all of the above tricks, we obtained results for the minimum identifying code problem on the $d = 2, n = 4$ undirected de Bruijn graph on the D-Wave hardware.

SAT formulation

Figure 25 shows the full satisfiability formulation of the 1-identifying code for $\mathcal{B}(2,4)$. This code was generated using a Matlab function written by Steve Adachi. It contains 50 clauses over 16 variables.

Problem Decomposition

For the full satisfiability formulation in Figure 25, the Ising model was too large to embed on the current D-Wave hardware graphs. The problem must be decomposed further. Note that two of the terms are $x_3 \vee x_{11}$ and $x_4 \vee x_{12}$. Thus, at least one of x_3 and x_{11} must be true, and at least one of x_4 and x_{12} must be true. Considering the branch where x_3 and x_4 are true, the satisfiability problem reduces to the problem shown in Figure 26. This decomposed formulation consists of just 24 clauses over 14 variables.

Satisfiability clause to Ising model mapping

Using the Ising model mappings shown in Figure 22, we generated an Ising model with 49 auxiliary variables $\{z_i\}$ for a total of 63 variables. We furthermore added the penalty term $\lambda \sum_i x_i$ so that the ground state will be a minimum 1-identifying code. Note that this is far better than the 1200+ auxiliary (slack) variables required in the integer program formulation for this case.

Figure 27 shows the logical graph of the Ising model. In the figure, nodes corresponding to the original 14 boolean variables are shown in green; the remaining nodes represent the auxiliary variables added during the satisfiability-to-Ising mapping process.

Embedding

Using the D-Wave embedding function `sapiFindEmbedding()`, we found an embedding of the Ising model onto the current (504-qubit) hardware graph for the Lockheed-Martin D-Wave machine that uses 253 physical qubits, with a maximum chain length of 8. This is shown in Figure 28.

In Figure 28, physical qubits corresponding to the same logical qubit have the same color and are labeled with the same number. The unlabeled red qubits are known faulty qubits and are not used.

Gauge Transformations

Since we have an exact solution from the parallel computing method on the graph $\mathcal{B}(2,4)$, we know that the $x_3 = x_4 = 1$ branch of the problem should have

```
( x2 v x3 )
( x0 v x2 v x4 v x5 v x8 v x9 )
( x0 v x3 v x6 v x7 v x8 v x9 )
( x0 v x1 v x2 v x4 v x9 v x10 )
( x4 v x12 )
( x0 v x1 v x6 v x9 v x12 v x14 )
( x0 v x3 v x4 v x5 v x8 v x9 )
( x0 v x2 v x6 v x7 v x8 v x9 )
( x0 v x1 v x3 v x4 v x9 v x10 )
( x1 v x5 v x8 v x10 )
( x1 v x4 v x9 v x10 v x11 )
( x0 v x2 v x5 v x8 v x9 v x12 )
( x1 v x3 v x5 v x12 )
( x1 v x2 v x9 v x10 v x13 )
( x1 v x6 v x9 v x11 v x14 v x15 )
( x1 v x3 v x7 v x8 v x12 v x14 )
( x1 v x3 v x6 v x9 v x14 v x15 )
( x4 v x5 v x8 v x9 v x11 )
( x0 v x1 v x2 v x9 v x10 v x12 )
( x3 v x8 v x10 v x12 )
( x2 v x5 v x8 v x9 v x13 )
( x2 v x4 v x11 v x13 )
( x2 v x6 v x7 v x10 v x13 )
( x2 v x5 v x6 v x13 v x14 )
( x3 v x5 v x7 v x12 )
( x3 v x10 v x12 v x14 )
( x3 v x5 v x6 v x13 v x14 v x15 )
( x3 v x6 v x7 v x10 v x13 v x15 )
( x3 v x11 )
( x0 v x1 v x4 v x6 v x9 v x14 )
( x4 v x6 v x7 v x10 v x11 )
( x4 v x5 v x6 v x11 v x14 )
( x5 v x7 v x10 v x14 )
( x5 v x6 v x11 v x12 v x14 v x15 )
( x5 v x6 v x11 v x13 v x14 v x15 )
( x6 v x7 v x8 v x9 v x13 v x15 )
( x6 v x7 v x8 v x9 v x12 v x15 )
( x6 v x7 v x10 v x11 v x12 v x15 )
( x6 v x7 v x10 v x11 v x13 v x15 )
( x12 v x13 )
( x0 v x1 v x8 )
( x1 v x2 v x4 v x5 v x9 )
( x1 v x3 v x6 v x7 v x9 )
( x2 v x4 v x8 v x9 v x10 )
( x2 v x5 v x10 v x11 )
( x4 v x5 v x10 v x13 )
( x5 v x6 v x7 v x11 v x13 )
( x6 v x8 v x9 v x12 v x14 )
( x6 v x10 v x11 v x13 v x14 )
( x7 v x14 v x15 )
```

Figure 25: Satisfiability Formulation for $\mathcal{B}(2,4)$

```
( x0 v x1 v x6 v x9 v x12 v x14 )
( x0 v x2 v x6 v x7 v x8 v x9 )
( x1 v x5 v x8 v x10 )
( x0 v x2 v x5 v x8 v x9 v x12 )
( x1 v x2 v x9 v x10 v x13 )
( x1 v x6 v x9 v x11 v x14 v x15 )
( x0 v x1 v x2 v x9 v x10 v x12 )
( x2 v x5 v x8 v x9 v x13 )
( x2 v x6 v x7 v x10 v x13 )
( x2 v x5 v x6 v x13 v x14 )
( x5 v x7 v x10 v x14 )
( x5 v x6 v x11 v x12 v x14 v x15 )
( x5 v x6 v x11 v x13 v x14 v x15 )
( x6 v x7 v x8 v x9 v x13 v x15 )
( x6 v x7 v x8 v x9 v x12 v x15 )
( x6 v x7 v x10 v x11 v x12 v x15 )
( x6 v x7 v x10 v x11 v x13 v x15 )
( x12 v x13 )
( x0 v x1 v x8 )
( x2 v x5 v x10 v x11 )
( x5 v x6 v x7 v x11 v x13 )
( x6 v x8 v x9 v x12 v x14 )
( x6 v x10 v x11 v x13 v x14 )
( x7 v x14 v x15 )
```

Figure 26: Decomposed Satisfiability Formulation for x_3, x_4 true.

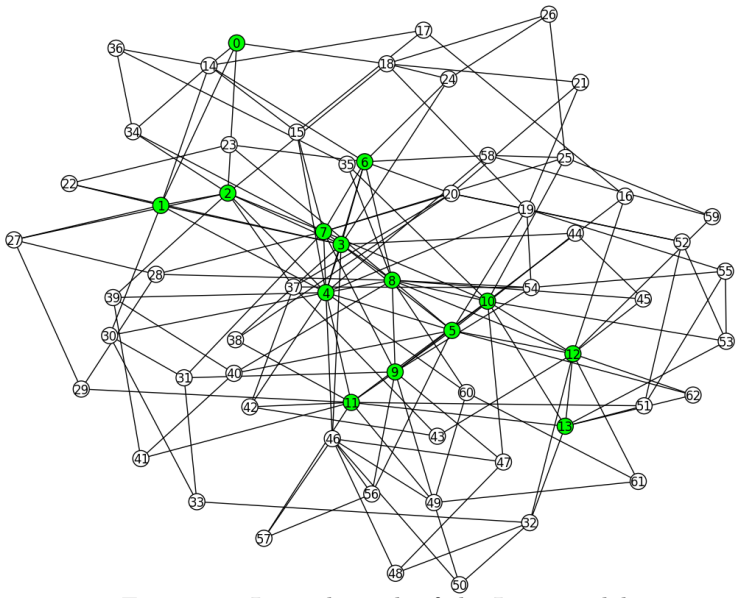


Figure 27: Logical graph of the Ising model

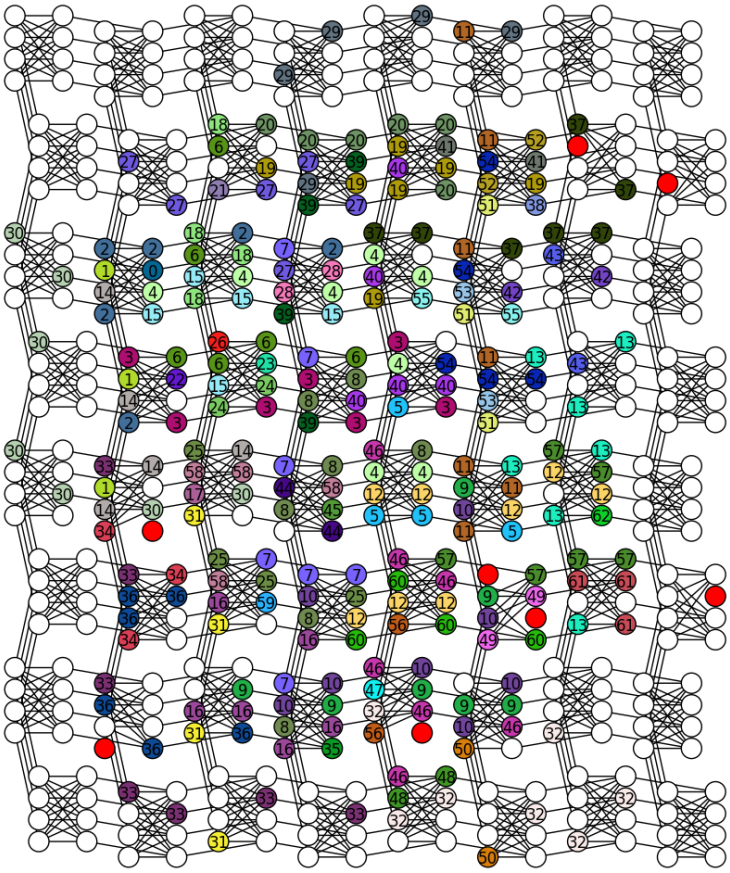


Figure 28: Embedding the problem onto the D-Wave hardware

a ground state with the remaining variables

$$x_8 = x_{10} = x_{13} = x_{14} = 1$$

corresponding to the minimum code length of 6. However, using the standard D-Wave embedding solver (which does not yet support gauge transformations - it uses all ferromagnetic couplings for the chains), we were not able to find this ground state. Even using the maximum allowed combinations of D-Wave function parameters `annealing_time` and `num_reads` (the number of annealing runs per call), the best that we could obtain were solutions corresponding to a code length of 7.

On the other hand, using a homegrown equivalent of the embedding solver, which also has the capability to incorporate gauge transformations, we were able to find the above ground state corresponding to a code of length 6.

4.6.3 Satisfiability Modulo Theory Solvers

Satisfiability Modulo Theory (SMT) is a current area of research that is concerned with the satisfiability of formulas with respect to some background theory [3]. These solvers combine boolean satisfiability solving with decision procedures for specific theories. For example, consider the following problem.

$$a = b + 1, \quad c > a, \quad c < b$$

In the theory of the integers, this problem is not satisfiable, however in the theory of the real numbers it is satisfiable. In general, solving an SMT problem consists of first solving a satisfiability problem, then doing theory-specific reasoning, and then possibly going back and changing the satisfiability problem. This process is repeated if necessary. In addition, multiple theories can also be used in the same satisfiability modulo theory problem instance, which may require additional repeats of this method.

To use these solvers on our identifying code problem for the undirected de Bruijn graph, we must first come up with a formulation of the problem using decision procedures. The graph $\mathcal{B}(d, n)$, contains d^n nodes. For each of these, we create a boolean variable that denotes whether or not the node is part of the identifying code. We then also create an array of boolean variables for that node's identifying set. An assertion is added to make sure that each element of the array is true if and only if the corresponding neighbor's boolean variable is true (i.e. if and only if the neighbor is part of the identifying code). To ensure unique codes, we add a statement to require that each node's identifying set is unique from every other node's identifying set. Then, to get codes of a fixed size, we create an integer variable for each node and add the constraints that the integer is at least 0 and no greater than 1. Next we add an assertion that each node's integer variable is 1 if and only if its boolean variable is true. Finally, we add a constraint that the sum of all of the integer variables is equal to the desired identifying code size.

$d \setminus n$	2	3	4	5	6	7
2	$\times$	4	6	12	(24)	(110)
3	4	9				
4	5	15				
5	6					
6	8					
7	9					
8	(10)					

Figure 29: Minimum 1-identifying codes on $\mathcal{B}(d, n)$

Now that the formulation of the problem has been determined, we can use a commercial solver to find solutions. For this work, we used the solver Z3, made by Microsoft Research. We begin by first picking a code length, and asking if there exists an identifying code of that length. If not, then the code length is increased by 1 and the problem is posed to Z3 again. This continues until an identifying code of a specific size is found. To find *all* satisfying models, after a single model was found an assertion is inserted into the formulation that requires that the the previously found identifying code be eliminated as an option. This forces Z3 to produce a different solution, or to state that the formulation is unsatisfiable (and hence no more identifying codes of that size exist). This process is repeated in a loop to obtain all identifying codes.

Using this approach on a single core, we were able to reproduce our results for $\mathcal{B}(d, n)$ from the parallel computing method in much less time. See Figure 29 for a summary of these results. The numbers in parentheses denote that we found a code of that size, but did not eliminate the possibility of a smaller code existing.

Because of the advancements in current satisfiability and satisfiability modulo theory solvers, they offer the potential to scale much better than a parallelized brute force approach. This is due in part to the fact that many of today's solvers are capable of realizing which subsets of assignments will define an unsatisfiable result, and hence they will avoid models in which those statements are set. In our problem, this might correspond to a case in which nodes A and B have the same identifying set. In this case, the solver would not bother looking at combinations of True/False assignments on the other nodes that do not affect the identifying sets of A or B.

In addition to the sophistication of today's solvers, there is also the possibility of parallelizing the search. While some instances were run manually in a parallel manner for this experiment, there is some research to be done on automatically parallelizing the search in order to further our known minimum results.

4.7 Minimum Identifying Code Examples

In this appendix, we give examples of some minimum identifying codes on de Bruijn networks. These identifying codes allow a vertex to be identified by the empty set.

4.7.1 Size of Minimum Identifying Codes

$\mathcal{B}(d,n)$	2	3	4	5
2	×	4	6	12
3	4	8		
4	4			
5	6			

4.7.2 Number of Minimum Identifying Codes

$\mathcal{B}(d,n)$	2	3	4	5
2	×	4	44	1694
3	3	156		
4	36			
5	500			

4.7.3 Complete Sets of Min. Identifying Codes

$\mathcal{B}(2,2)$**:** Since $\{01, 10\}$ are twin vertices, no identifying code is possible.

$\mathcal{B}(3,2)$**:** The minimum identifying codes have size 4.

- $\{01, 02, 10, 20\}$
- $\{01, 10, 12, 21\}$
- $\{02, 12, 20, 21\}$

$\mathcal{B}(4,2)$**:** The minimum identifying codes have size 4.

- $\{01, 02, 10, 20\}$
- $\{01, 02, 10, 30\}$
- $\{01, 02, 20, 30\}$
- $\{01, 03, 10, 20\}$
- $\{01, 03, 10, 30\}$
- $\{01, 03, 20, 30\}$
- $\{01, 10, 12, 21\}$
- $\{01, 10, 12, 31\}$
- $\{01, 10, 13, 21\}$
- $\{01, 10, 13, 31\}$

- $\{01, 12, 13, 21\}$
- $\{01, 12, 13, 31\}$
- $\{02, 03, 10, 20\}$
- $\{02, 03, 10, 30\}$
- $\{02, 03, 20, 30\}$
- $\{02, 12, 20, 21\}$
- $\{02, 12, 20, 23\}$
- $\{02, 12, 21, 23\}$
- $\{02, 20, 21, 32\}$
- $\{02, 20, 23, 32\}$
- $\{02, 21, 23, 32\}$
- $\{03, 13, 30, 31\}$
- $\{03, 13, 30, 32\}$
- $\{03, 13, 31, 32\}$
- $\{03, 23, 30, 31\}$
- $\{03, 23, 30, 32\}$
- $\{03, 23, 31, 32\}$
- $\{10, 12, 21, 31\}$
- $\{10, 13, 21, 31\}$
- $\{12, 13, 21, 31\}$
- $\{12, 20, 21, 32\}$
- $\{12, 20, 23, 32\}$
- $\{12, 21, 23, 32\}$
- $\{13, 23, 30, 31\}$
- $\{13, 23, 30, 32\}$
- $\{13, 23, 31, 32\}$

$\mathcal{B}(2, 3)$**:** The minimum identifying codes have size 4.

- $\{001, 010, 011, 101\}$
- $\{001, 010, 101, 110\}$
- $\{010, 011, 100, 101\}$
- $\{010, 100, 101, 110\}$

Conclusion

At the conclusion of this effort we have obtained new results, bounds, and algorithms for t-identifying codes on $\vec{\mathcal{B}}(d, n)$, however there is still room for improvement. In particular, the problem of finding t-identifying codes in the undirected de Bruijn graphs remains an open and challenging problem of interest.

For future efforts, we suggest the following two avenues. First, we aim to continue analyzing de Bruijn graphs and their internal structures that have proven useful in many applications. This includes continuation of our current exploration of identifying codes and expanding to consider other related graph structures, such as robust identifying codes that are resilient against node and link failure. We will continue our quest for constructions of optimal identifying codes in the undirected de Bruijn networks as well as consider approximation algorithms. This will include examining existing algorithms for identifying codes and modifying the methods to take advantage of the de Bruijn graph properties. Additional key vertex subsets will also be considered, such as resolving sets and locating dominating sets.

Our second research direction will be to consider variations on de Bruijn networks and perform similar analyses. Traditional de Bruijn networks are based on strings over a fixed alphabet, and variations that have yet to be examined are based on different combinatorial objects such as permutations. A different variation on de Bruijn networks, known as alphabet overlap graphs, provide a much denser, more highly connected variant of the de Bruijn graph. These graphs are relative newcomers to the academic arena, so a complete analysis of their structural properties is needed to determine their relevance and applicability to real-world networks.

Bibliography

[1] Toru Araki. On the k-tuple domination of de bruijn and kautz digraphs. Inform. Process. Lett., 104(3):86–90, 2007.

[2] Joel Baker. De bruijn graphs and their applications to fault tolerant networks. Master's thesis, Oregon State University, 2011.

[3] C. Barrett, R. Sebastiani, S.A. Seshia, and C. Tinelli. Handbook of Satisfiability, volume 185 of Frontiers in Artificial Intelligence and Applications, chapter 26: Satisfiability Modulo Theories, pages 825–885. IOS Press, February 2009.

[4] J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. Theoret. Comput. Sci., 218:135–141, 1999.

[5] Uri Blass, Iiro Honkala, and Simon Litsyn. Bounds on identifying codes. Discrete Math., 241(1-3):119–128, 2001. Selected papers in honor of Helge Tverberg.

[6] Debra L. Boutin. Identifying graph automorphisms using determining sets. Electron. J. Combin., 13(1):R78, 2006.

[7] Irène Charon, Iiro Honkala, Olivier Hudry, and Antoine Lobstein. Structural properties of twin-free graphs. Electron. J. of Combin., 14(1):R16, 2007.

[8] Gary Chartrand, Michael Raines, and Ping Zhang. The directed distance dimension of oriented graphs. Math. Bohem., 125(2):155–168, 2000.

[9] D-Wave: The Quantum Computing Company. Programming with qubos. Instructional Document, 2013.

[10] N.J. Fine and H.S. Wilf. Uniqueness theorems for periodic functions. Proc. Amer. Math. Soc., 16(1):109–114, Feb., 1965.

[11] F. Foucaud, S. Gravier, R. Naserasr, A. Parreau, and P. Valicov. Identifying codes in line graphs. J. of Graph Theory, 73:425–448, 2013.

[12] Florent Foucaud, George B. Mertzios, Reza Naserasr, and Aline Parreau. Identification, location-domination and metric dimension on interval and permutation graphs: I. bounds. preprint.

[13] R. L. Graham, M. Grötschel, and L. Lovász, editors. Handbook of combinatorics. Vol. 1, 2. Elsevier Science B.V., Amsterdam; MIT Press, Cambridge, MA, 1995.

[14] Mark G. Karpovsky, Krishnendu Chakrabarty, and Lev B. Levitin. On a new class of codes for identifying vertices in graphs. IEEE Trans. Inform. Theory, 44(2):599–611, 1998.

[15] Y. Kikuchi and Y. Shibata. On the domination numbers of generalized de bruijn digraphs and generalized kautz digraphs. Inform. Process. Lett., 86:79–85, 2003.

[16] D.L. Kreher and D.R. Stinson. Combinatorial Algorithms: Generation, Enumerations, and Search, pages 825–885. CRC Press, 1998.

[17] Zhen Liu. Optimal routing in the De Bruijn networks. Research Report RR-1130, Inria, 1990.

[18] Marilynn Livingston and Quentin F. Stout. Perfect dominating sets. Congr. Numer., 79:187–203, 1990.

[19] Richard P. Stanley. Enumerative combinatorics. Volume 1, volume 49 of Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, second edition, 2012.

[20] Yi-Chun Xu and Ren-Bin Xiao. Identifying code for directed graph. In Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing - Volume 02, SNPD '07, pages 97–101, Washington, DC, USA, 2007. IEEE Computer Society.